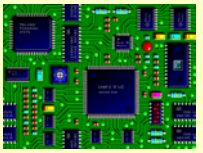
Conception et simulation des systèmes

VHDL (Langage et Modélisation)



Université de Rouen
GPM (UMR CNRS 6634) 76801 Saint-Etienne du Rouvray Cedex
Richard GRISEL, Professeur des Universités
Richard.Grisel@univ-rouen.fr

Avant - propos

Ces transparents correspondent à un cours VHDL, commencé à l 'ICPI (Institut de Chimie et Physique Industrielle de Lyon devenue ESCPE après la fusion entre l 'ESCIL et l 'ICPI) en 1992. Ce cours a ensuite évolué pour faire partie du module optionnel « VHDL » du DEA Dispositifs de l 'Électronique Intégrée (École Doctorale SPI-EEA de Lyon).

Il a aussi été initié pour les étudiants de l 'Option micro-électronique de l 'INSA de Lyon (Département GE).

Il a été dispensé à Amiens aux étudiants de l 'Option AII (Automatique et Informatique Industrielle) de l 'IUP GEII au niveau maîtrise. Il est précédé d 'une introduction aux différents types de circuits intégrés.

Il est aussi utilisé en Master 2, option OSM de l'Université de Rouen et a été enseigné au département ASI de l'INSA de Rouen

En fonction du type d 'auditoire, ce cours (+ quelques exercices qui ne figurent pas dans la version électronique) peut prendre dans sa version la plus complète une vingtaine d 'heures, et dans sa version réduite une dizaine d 'heures.

Il a aussi servi de trame VHDL pour une mission d 'expertise pour les nations unies en Chine entre 1993 et 1996 pour l 'enseignement plus général de la CAO des systèmes électronique mais avec une adaptation aux supports en anglais fournis par ViewLogic France.

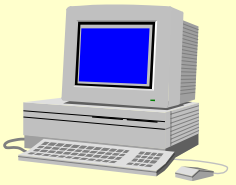
Je suis à votre écoute pour toute remarque, « add-ons », permettant de continuer à le faire évoluer.

R. Grisel, Rouen, le 17 octobre 2007

Modélisation Matérielle (ModelWare)

- Modélisation "Ingénierie Matérielle"

- Modélisation "Ingénierie Logicielle"



Modélisation = Ingénierie Matérielle

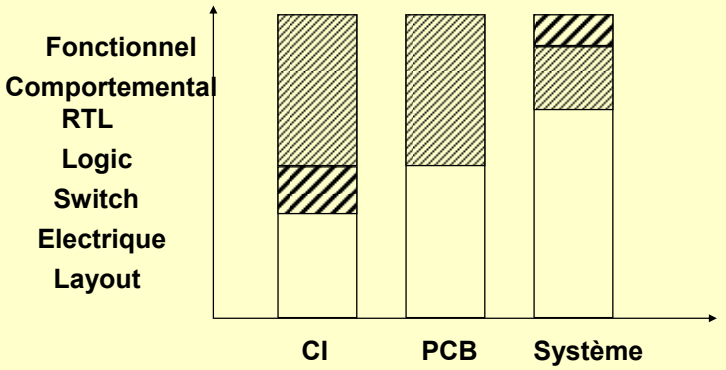
Un modèle peut être utilisé pour :

- Conception "Top-Down" (Spécifications à haut niveau jusqu'à implémentation détaillée

- Conception "Bottom-Up" (Assemblage à partir d'une librairie existante)

- Conception mixte ("Meet in the Middle")

Niveaux de description en VHDL

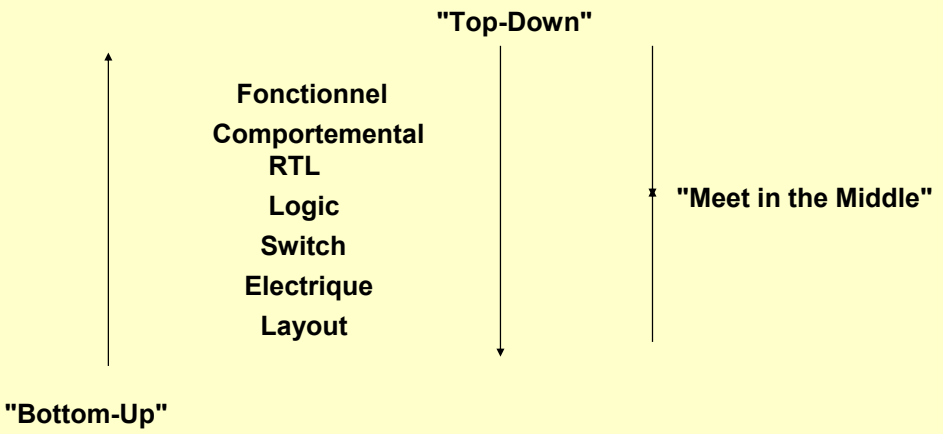


Complètement supporté

Possible mais pas optimal

Non supporté

Methodologies de Conception



Modélisation = Ingénierie logicielle

Un modèle est (supposé être) exécutable

Une conception peut concerner plusieurs ingénieurs

Une conception "grandeur réelle" peut correspondre à plus de 50000 lignes de code VHDL

Modélisation = Communication

Un modèle doit :

- refléter l'intention du concepteur

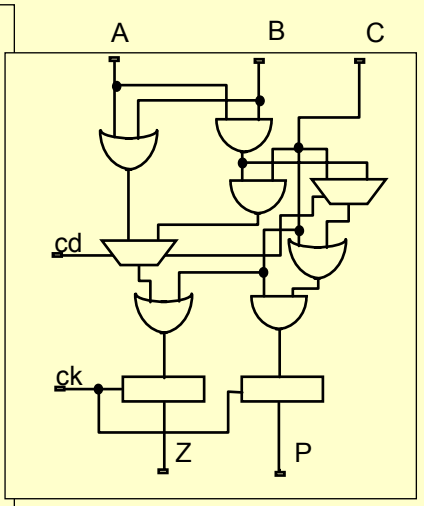
- permettre une utilisation intra(inter)-entreprise
  - utilisation comme spécification (par un concepteur)
  - simulable par le client

- permettre une (re)conception à n'importe quelle date

- montrer ou cacher le savoir-faire ("Know-how") de l'entreprise

VHDL pour la synthèse

```
entity cond1 is
port(A,B,C : in Bit_vector(1 to 5);cd,ck:boolean;
Z,P: out Bit_vector(1 to 5))
end cond1
architecture behaviour of cond1 is
begin  process
   variable temp, t2 : Bit_vector(1 to 5)
constant n:integer:=6;
begin wait until clk
t2:=c;temp:=(A and B) nand C;
if cd then temp:=a or b; t2:=a and b; end if;
t2:= t2 xor C; z<=temp or C; P<=t2 and C;
end process;
end behaviour;
```



Cours VHDL Médiathèque e-EEA - Richard Grisel – GPM (UMR CNRS 6634) Université de Rouen - Richard.Grisel@univ-rouen.fr

Le manuel de référence du langage (LRM)

Le "LRM" (Language Reference Manual) est la bible "sémantique" de VHDL

Très difficile à lire (sauf chap. 1 : design entities)

A utiliser via le Glossaire, l'Index et chap. 14 (predefined environment)

Les messages d'erreur font souvent référence à des paragraphes du LRM (paragraph #)

Bibliographie

The VHDL Handbook (Coehlo, Kluwer)

Chip Level Modelling in VHDL (Armstrong, prenctice Hall)

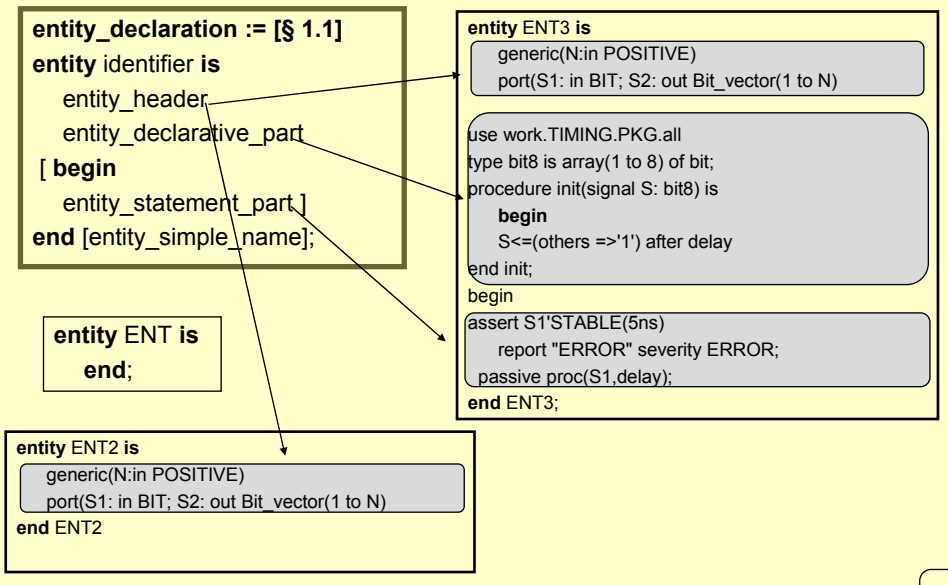
VHDL Tutorial (CLSI)

VHDL : Hardware description and design (Lipsett, Schaefer, Ussery, Kluwer)

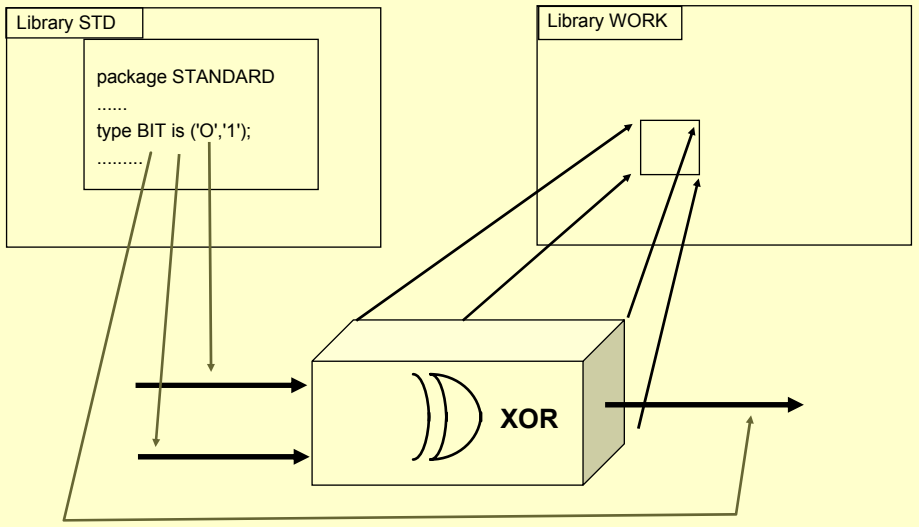
VHDL: du langage à la modélisation (Airiau, Bergé, Olive, Rouillard, Presses Univ. Romandes)

VHDL Designer's reference (Bergé, Maginot, Fonkoua, Rouillard, Kluwer)

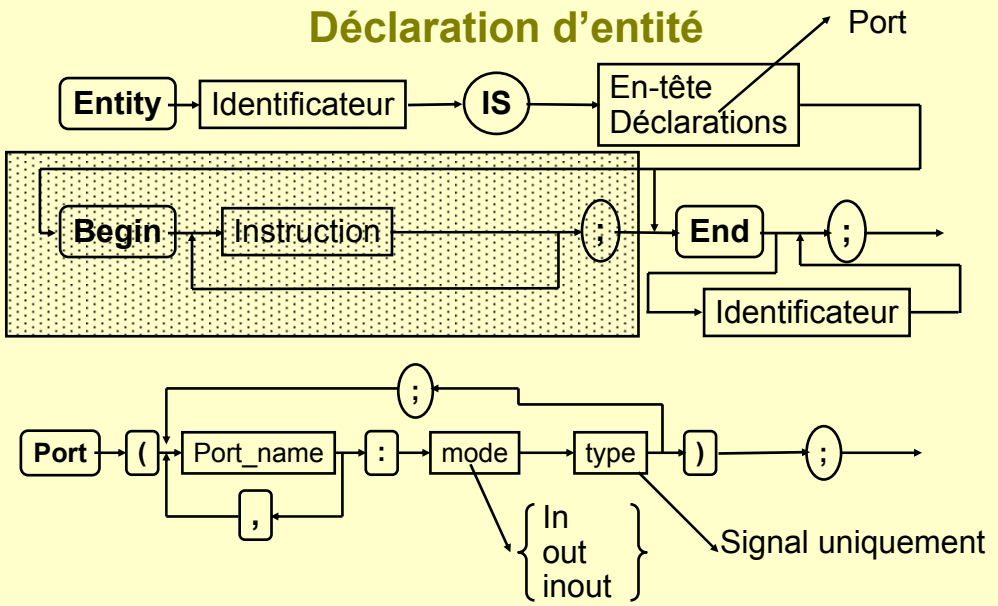
La Grammaire VHDL



VHDL : Première porte : Porte XOR

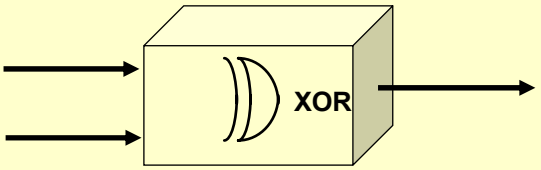


Déclaration d'entité



Porte XOR : Vue Externe

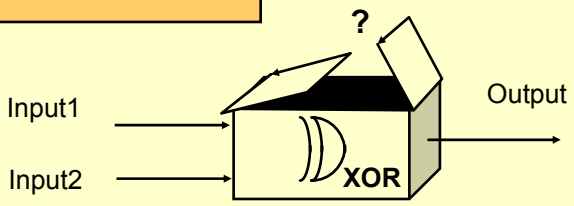
```
entity XOR_ENT is
 port(Input1, Input2 : in BIT; Output : out BIT);
end XOR_ENT;
```



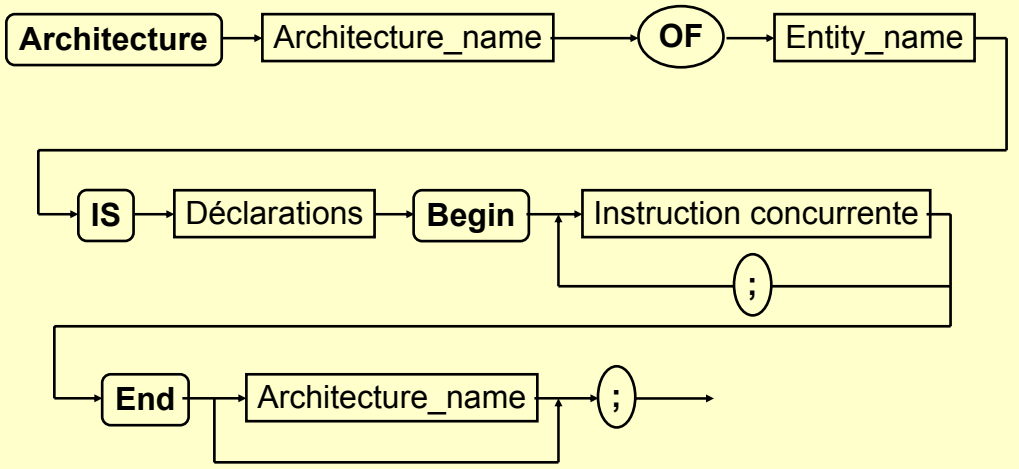
Vue Externe = Boite Noire

```
entity XOR_ENT is
 port(Input1, Input2 : in BIT; Output : out BIT);
end XOR_ENT;
```

```
architecture STRUCTURAL of XOR_ENT is
 begin

   ??

end STRUCTURAL;
```

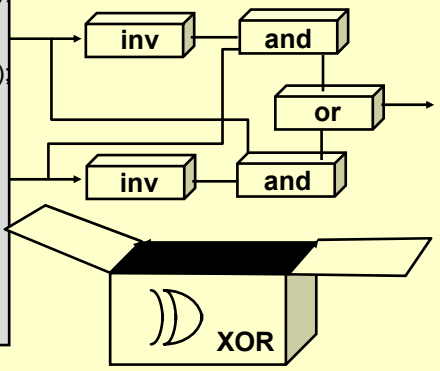


Déclaration d'Architecture

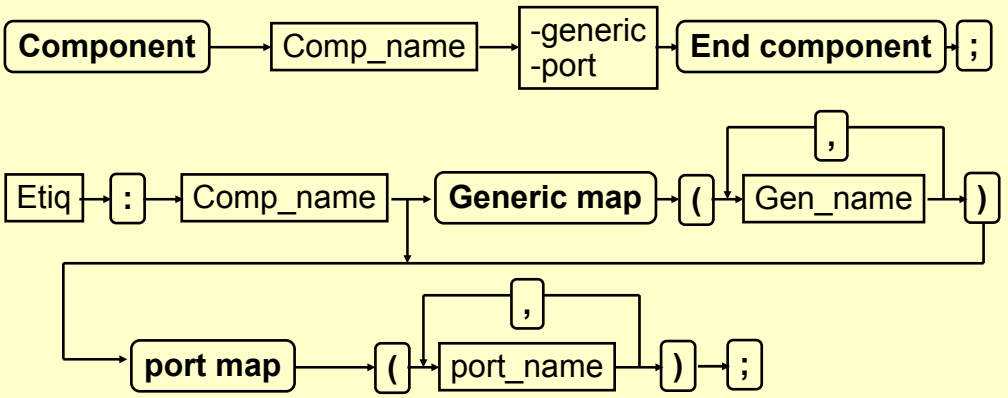


A l' intérieur de la boite

```
architecture STRUCTURAL of XOR_ENT is
 component INVCMP
    port (Input: in BIT; Output : out BIT);
 end component;
 component ANDCMP
    port (Input1, Input2 : in BIT; Output :out BIT);
 end component;
 component ORCMP
    port (Input1, Input2 : in BIT; Output :out BIT);
 end component;
signal S1, S2, S3, S4 : BIT;
begin
    INV1 : INVCMP port map (Input1, S1);
    INV2 : INVCMP port map (Input2, S2);
    AND1 : ANDCMP port map (S1, Input2, S3);
    AND2 : ANDCMP port map (S2, Input1, S4);
    OR1 : ORCMP port map (S3, S4, Output);
end STRUCTURAL;
```

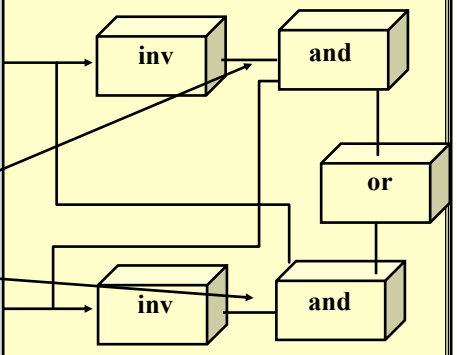


Déclaration et Instantiation de composant



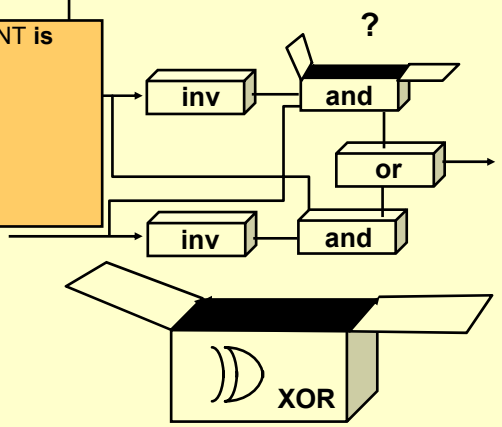
Signaux internes (Signals)

```
architecture STRUCTURAL of XOR_ENT is
 component INVCMP
    port (Input: in BIT; Output : out BIT);
 end component;
 component ANDCMP
    port (Input1, Input2 : in BIT; Output :out BIT);
 end component;
 component ORCMP
    port (Input1, Input2 : in BIT; Output :out BIT);
 end component;
 signal S1, S2, S3, S4 : BIT;
begin
    INV1 : INVCMP port map (Input1, S1);
    INV2 : INVCMP port map (Input2, S2);
    AND1 : ANDCMP port map (S1, Input2, S3);
    AND2 : ANDCMP port map (S2, Input1, S4);
    OR1 : ORCMP port map (S3, S4, Output);
end STRUCTURAL;
```



A l'intérieur de la boite

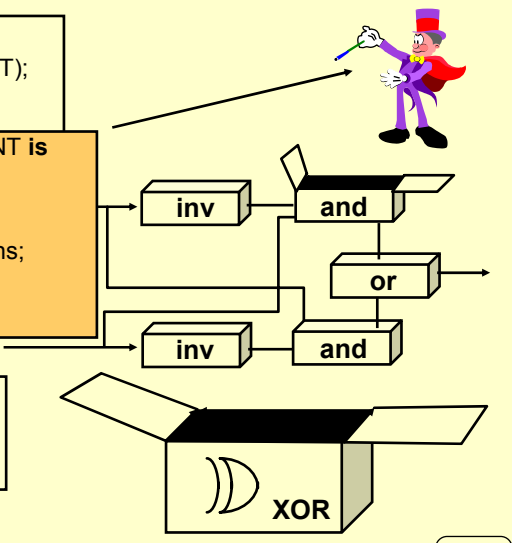
```
entity AND_ENT is
 port(Input1, Input2 : in BIT; Output : out BIT);
end AND_ENT;

architecture DATA_FLOW of AND_ENT is
 begin

   ?

end DATA_FLOW;
```



Description "Data-Flow" (1)

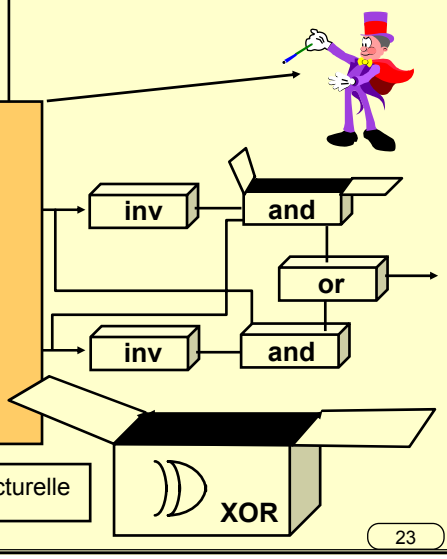
```
entity AND_ENT is
 port(Input1, Input2 : in BIT; Output : out BIT);
end AND_ENT;

architecture DATA_FLOW of AND_ENT is
 begin

   Ouput <= Input1 and input2 after 5 ns;

end DATA_FLOW;
```

A chaque changement de Input1 ou de Input2,
changement de la sortie après 5 ns



Description comportementale (2)

```
entity AND_ENT is
 port(Input1, Input2 : in BIT; Output : out BIT);
end AND_ENT;

 architecture BEHAVIOUR of AND_ENT is
 begin
process
begin   if (Input1='1' and Input2='1')
           then Output <= '1'
           else Output <= '0';
           end if;
wait on Input1, Input2;
end process;
end BEHAVIOUR;
```

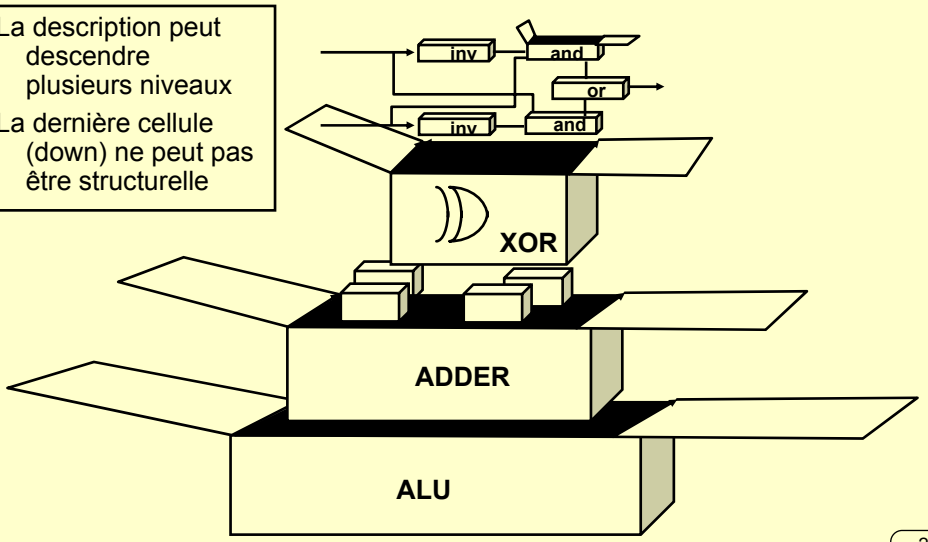


L'architecture n'a pas de correspondance structurelle

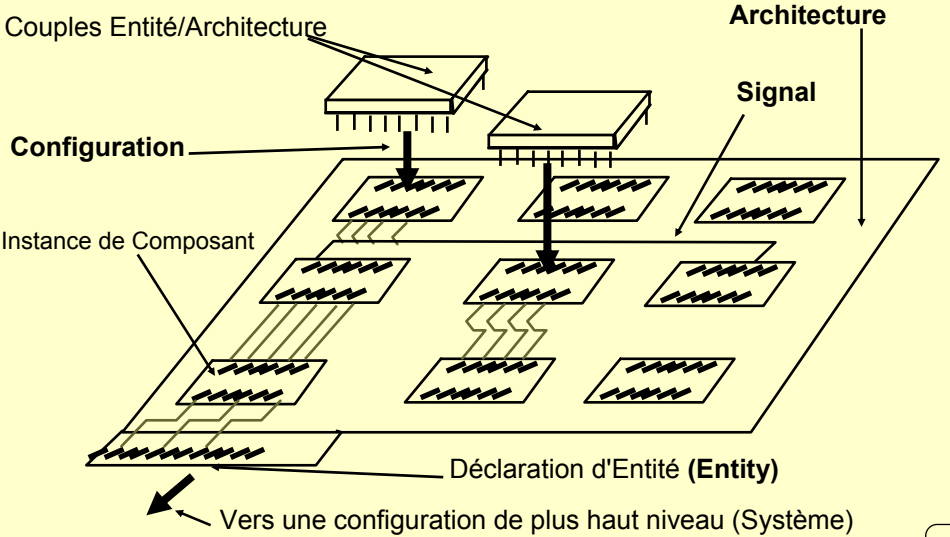
Description Structurelle (3)

La description peut descendre plusieurs niveaux

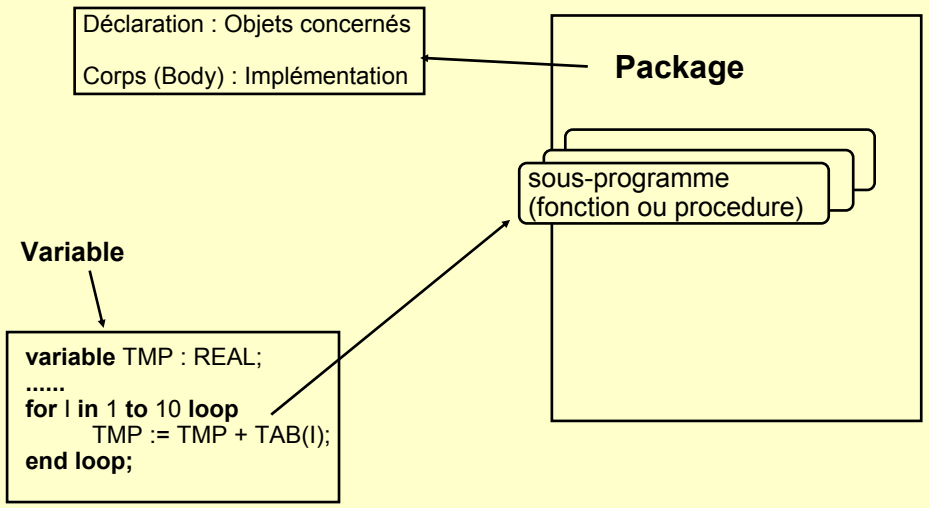
La dernière cellule (down) ne peut pas être structurelle



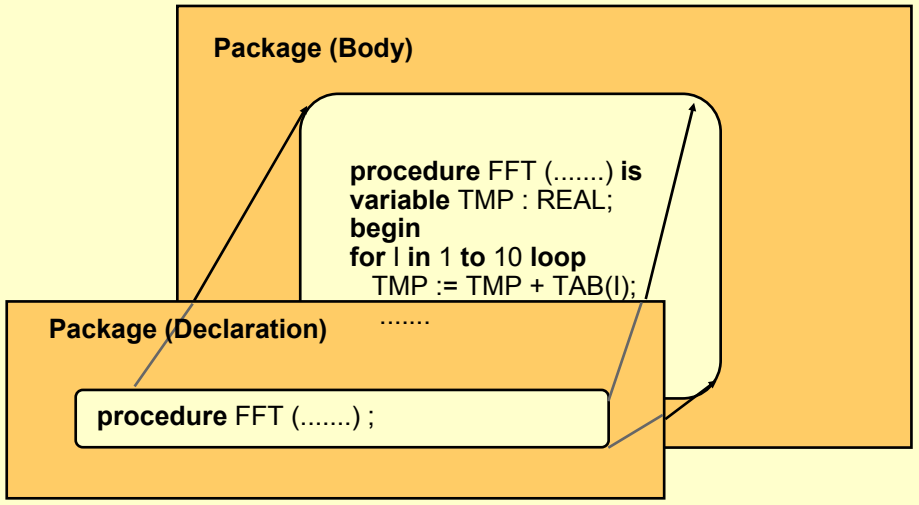
Vue d'ensemble d'une description VHDL (Hiérarchie structurelle)



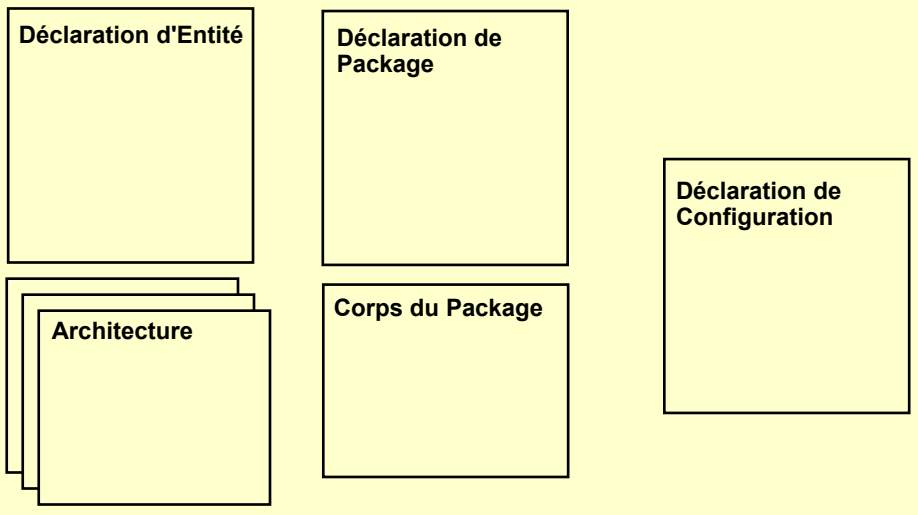
Vue d'ensemble d'une description VHDL (Hiérarchie Algorithmique - 1)



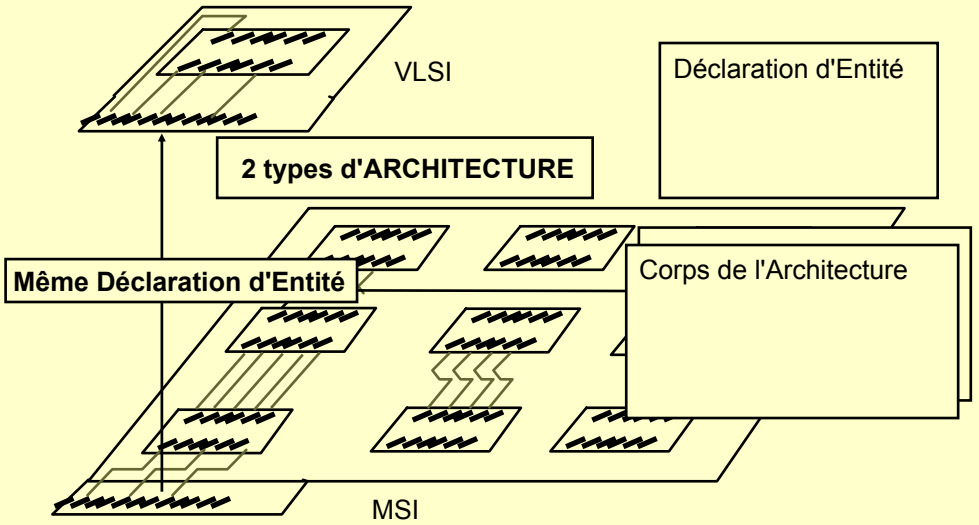
Vue d'ensemble d'une description VHDL (Hiérarchie Algorithmique - 2)



Unités de conception VHDL



Architectures Multiples



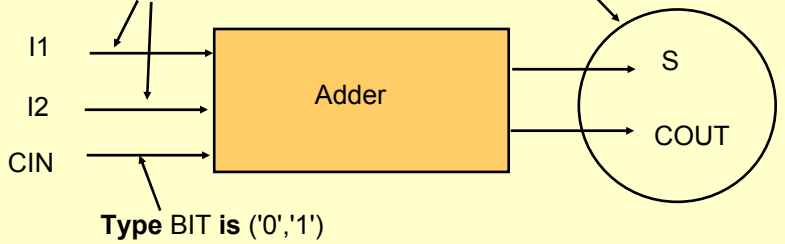
Lexique

- **Identificateurs:** FOO, FOO_2, FOO_UN ~~(_FOO, FONC_, FO__UN, 2FU, FX$)~~
- **Mots Clés:** abs, access, after
- **Délimiteurs:** +, -, *, =, <=
- **Entiers littéraux:** 2, 3E4, 100_000 ~~(2.0, 3.0E4, _1, 45_)~~
- **Littéraux basés:** 8#7327720#, 16#AC3E#, 2#10001# ~~(17#ABCD#)~~
- **Chaines de bits:** "0001_0101_1111", O"12753", X"2FAC" ~~("0112_0101_1110")~~
- **Chaînes de caractères:** 'A', '', CR, LF
- **Littéraux Physiques:** 10 ns, 25 pf, 4.7 kOhm ~~(45ns)~~
- **Réels:** 2.3, 0.1, 1.4E5, 1.2E-2, 1.000_002 ~~(.3, 3., 2)~~
- **Commentaires:** texte -- Commentaire à fin de ligne

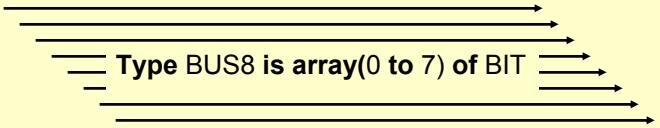
Types (1)

Type EXTENSION **is record**
Valeur : ENTIER;
Retenue : BIT;
end record

Type ENTIER **is range** 0 **to** 255



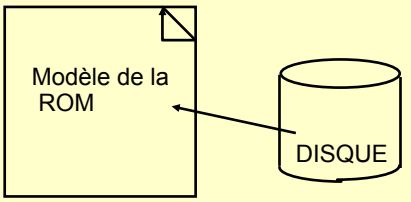
Type BIT **is** ('0','1')

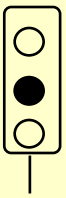


Type BUS8 **is array**(0 **to** 7) **of** BIT

Types (2)

type Fichier_type **is file of** INTEGER;



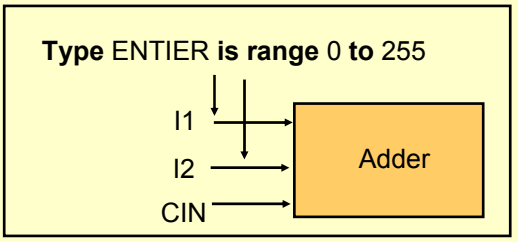


type Adresse **is access** NODE;

Type STATE **is** (RED, ORANGE, GREEN)

Type FSMSTATE **is** (DESCENTE, MONTEE, ARRET)

Types de base



Numériques : INTEGER, REAL

123, 1_123, 23E4, 0 sont des INTEGER

1_23.0, 2.3, 2.3E4, 0.0 sont des REAL

Pas de conversion implicite. Mais la conversion explicite est toujours possible :

Cette_valeur_reelle := REAL(Cet_integer)

Sous-programmes prédéfinis

Opérateurs relationnels :
"=", ">", "<", ">=", "<=", "/="

Opérateurs Arithmétiques :
"+", "*", "/", "-", "**", "mod", "abs", "rem"

Types énumérés

- **type** FEUX **is** (ROUGE, ORANGE, VERT);
- **type** ENUMERE **is** (UN, DEUX, TROIS);
- **type** CHARACTER **is** (....., 'A', 'B',.....);
- **type** BIT **is** ('0', '1');
- **type** VLBIT **is** ('X', 'Z', '0', '1',);
- **type** STD_ULOGIC **is** ('U', 'X', '0','1','Z','W', 'L','H','-');
- **type** MELANGE **is** (UN, '2', '3', QUATRE)

La même expression littérale peut correspondre à plusieurs types énumérés **(surcharge)**.

Sous-programmes prédéfinis :
Opérateurs relationnels : "=", ">", "<", ">=", "<=", "/="
Pour BIT et BOOLEAN : and, or, nand, nor, xor, not

Attributs Prédéfinis pour les types énumérés

SYNTAXE : **Identificateur** → **'** → **Attribut**

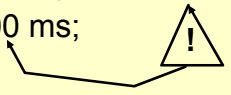
- **type** FEUX **is** (ROUGE, ORANGE, VERT);

FEUX'LEFT = ROUGE
FEUX'RIGHT = VERT
FEUX'SUCC(ROUGE) = ORANGE
FEUX'SUCC(VERT) Impossible (erreur)
FEUX'PRED(ORANGE) = ROUGE
FEUX'VAL(0) = ROUGE
FEUX'POS(VERT) = 2

Types Physiques (Physical types)

type TIME **is range** *somerange*
units
fs;
ps= 1000 fs;
ns= 1000 ps;
ms= 1000 ns;
sec= 1000 ms;
.....
end units;

Dépend de la machine (minimum 32 bits)



Sous-programmes prédéfinis

Opérateurs relationnels :
"=", ">", "<", ">=", "<=", "/="

Opérateurs Arithmétiques :
"+", "*", "/", "-","abs"

variable heure_de_la_pause : TIME;
.....
heure_de_la_pause := NOW + 2 hr + 25 min;

Types Physiques (Physical types)

Quelques sous-programmes (types mixtes) : time/time => integer
time/integer => time

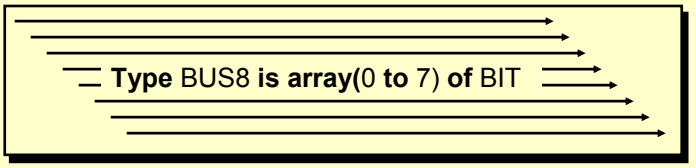
	gauche	**droite**	**résultat**
"*"	physique	integer	physique
	physique	real	physique
	integer	physique	physique
	real	physique	
"/"	physique	integer	physique
	physique	real	physique
	physique	même type	**integer**

Types Physiques (Physical types)

- Multiplication de P (type Tp) par I (Integer) :
  Tp'val(Tp'pos(P) * I) (unité de base du type)
- Multiplication de P (type Tp) par F (real) :
  Tp'val(INTEGER(REAL(Tp'pos(P))*F))
- Division de P (type Tp) par I (Integer) :
  Tp'val(Tp'pos(P) / I)
- Division de P (type Tp) par F (real) :
  Tp'val(INTEGER(REAL(Tp'pos(P))/F))
- Division de P1 (type Tp) par P2 (type Tp) :
  Tp'pos(P1) / Tp'pos(P2)

Ex ==> 345 fs + (347 ps / 20) + 2 ns vaut 2_017_695 fs
pour avoir le résultat en ns (---------) / 1 ns

Types tableaux (array)



type VECTEUR **is array** (1 **to** 10) **of** INTEGER;
type MATRICE **is array** (1 **to** 10, 1 **to** 10) **of** REAL;
type DATABUS **is array** (15 **downto** 0) **of** BIT;
type Table_de_verite **is array** (BIT,BIT) **of** BIT;
-- Package STD (prédéfinis)
type bit_vector **is array** (natural range <>) **of** bit
type string **is array** (natural range <>) **of char**

Attributs prédéfinis sur les types tableaux et sur les objets de type tableaux

T'range	dimension du tableau
T'reverse_range	dimension dans le sens inverse de la déclaration
T'low	valeur la plus faible
T'high	valeur la plus forte
T'left	valeur la plus à gauche
T'right	valeur la plus à droite
T'length	longueur du tableau

Attributs prédéfinis sur les types tableaux et sur les objets de type tableaux

```
type UN is array (15 downto 0) of BIT;
type DEUX is array (0 to 15) of BIT;
type TROIS is array (0 to 10, BOOLEAN) of BIT;
signal S: DEUX;
```

UN'RANGE => 15 downto 0
TROIS'RANGE(1) => 0 to 10
TROIS'RANGE(2) => FALSE to TRUE
S'RANGE => 0 to 15

S'LENGTH => 16

Objet S : STRING -----ERROR
S: STRING(1 to 10) --- OK

UN'LOW => 0, UN'LEFT => 15
DEUX'LOW => 0 , DEUX'LEFT => 0

Tableaux non contraints (déclaration de type)

type STRING **is array** (POSITIVE **range** <>) **of** CHARACTER; ------ type prédéfini

peut être utilisé :
* comme <<types de types >> pour générer des sous-types contraints :
subtype STRING80 **is** STRING (1 **to** 80);

* comme <<paramètre formel général >> pour les sous-programmes :

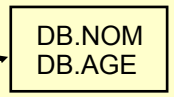
```
function MAJUSCULE(ARG:STRING) return STRING is
variable TMP : STRING (ARG'RANGE):=ARG;
constant GAP : NATURAL := CHARACTER'POS('a') - CHARACTER'POS('A');
  begin
  for I in TMP'RANGE          loop   if TMP(I) in 'a' to 'z' then
            TMP(I) := CHARACTER'VAL(CHARACTER'POS(ARG(I)) - GAP);
                                  end if;
                            end loop;
  return TMP;
end MAJUSCULE;
```

Objet S : STRING -----ERROR
S: STRING(1 to 10) --- OK

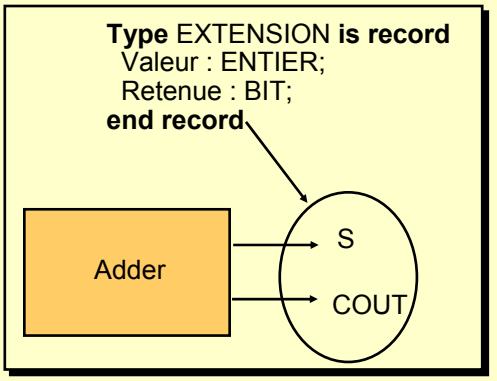
Type enregistrements (Record)

```
type IDENTITE is record
NOM : STRING(1 to 20);
AGE : INTEGER;
end record;

type RECURSIF;
type POINTE is access
   RECURSIVE;
type RECURSIF is record
   OBJET : SOME_TYPE;
   NEXT : POINTE;
end record;
```

Listes (New, Deallocate)



Déclaration anticipée



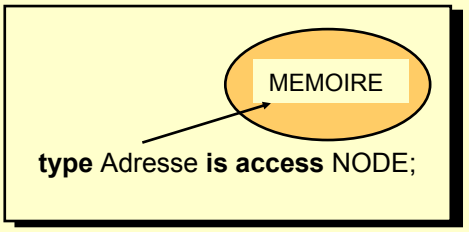
Types Pointeurs (Access)

Equivalent aux pointeurs PASCAL, C

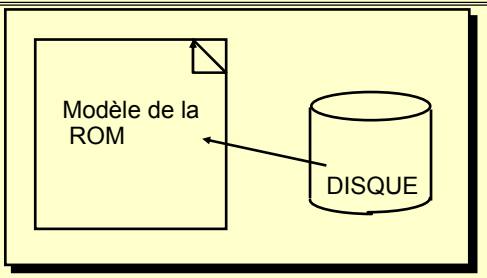
type TPE **is** .....any_type.....

type POINTE **is access** TPE;

variable V: PT;
...
V := **new** TPE; ---- allocation
V.all := objet de type TPE;
Note : mémoire non "hardware"



Types fichiers



type FICHIER **is file of** some_type;

...

file FICH : FICHIER **is out** "nom.ext";

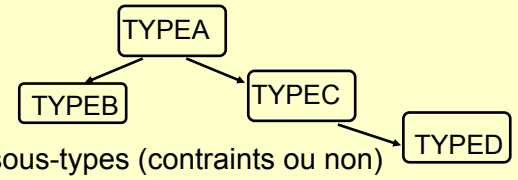
...

procédures READ, WRITE, ENDFILE (prédéfinies)

Ouverture automatique à l'élaboration

Fermeture automatique à la fin du programme d'utilisation du fichier

Sous-types



Construits à partir de types ou sous-types (contraints ou non)

Permettent de contraindre les types de base

subtype POSITIVE **is** INTEGER **range** 1 **to** INTEGER'LAST;

subtype STRING5 **is** STRING(1 **to** 5);

subtype DYNAMIC **is** SOME_TYPE (SOME_PARAMETER **downto** 1000);

Les sous-types sont compatibles avec
- types parents
- autres sous-types du type parent
- leurs propres sous-types

Ne définissent pas d'opérations mais héritent de celles des parents.

Les tests aux limites sont faits au "Run-time" (flexible mais coûteux en temps)

Sous-types résolus

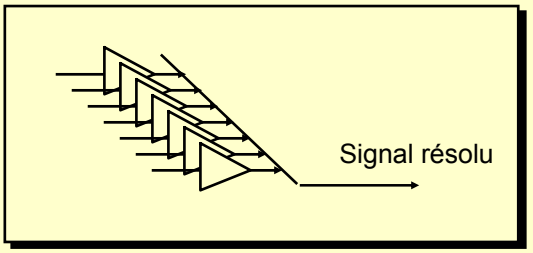
type BIT **is** ('0','1'); ----- prédéfini

type BIT_VECTOR **is array** (POSITIVE **range** <>) **of** BIT; --- prédéfini

fonction de résolution

function FN (ARG : **in** BIT_VECTOR) **return** BIT;

subtype RESOLVED **is** FN BIT;
signal A,B : BIT;
signal S1 : RESOLVED;
..
S1 <= A;
..
S1 <=B;



Fonction de résolution

```
function resolved ( s : std_ulogic_vector ) return std_ulogic;
variable result : std_ulogic := 'Z';
begin – le  test pour un pilote unique  est obligatoire sinon la boucle donnera  'X' pour un non initialisé à  '-'
 if s'length = 1 then return s(s'low);
else for i in s'range loop
result := resolution_table(result, s(i));
end loop;
end if;
return result;
end resolved;
```

Table de « résolution »

```
constant resolution_table : stdlogic_table :=
( -- ------------------------
--    U   X    0   1   Z   W   L   H   –          | |
-- ------------------------
    ( 'U', 'U', 'U', 'U', 'U', 'U', 'U', 'U', 'U' ),   -- | U |
    ( 'U', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X' ),   -- | X |
    ( 'U', 'X', '0', 'X', '0', '0', '0', '0', 'X' ),    -- | 0 |
    ( 'U', 'X', 'X', '1', '1', '1', '1', '1', 'X' ),    -- | 1 |
    ( 'U', 'X', '0', '1', 'Z', 'W', 'L', 'H', 'X' ),    -- | Z |
    ( 'U', 'X', '0', '1', 'W', 'W', 'W', 'W', 'X' ),    -- | W |
    ( 'U', 'X', '0', '1', 'L', 'W', 'L', 'W', 'X' ),    -- | L |
    ( 'U', 'X', '0', '1', 'H', 'W', 'W', 'H', 'X' ),    -- | H |
    ( 'U', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X' )     -- | - | );
```


Classes d'objets (1)

CONSTANTES :

évaluées une seule fois (partie déclarative), ne peuvent pas être changées

constant PI : REAL := 3.1415;

VARIABLES :

Uniquement déclarées dans le domaine séquentiel (process, sous-programmes). En conséquence, ne sont pas accessibles du domaine concurrent.

Une variable contient une seule valeur. La valeur peut changer sans effet de bord sur les parties qui ne la connaissent pas.

variable COMPTE : INTEGER;

Classes d'objets (2)

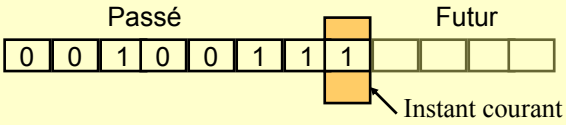
SIGNAUX (Supports de la simulation VHDL, liens avec le matériel) :

Un signal contient une valeur actuelle, un ensemble de valeurs passées, un ensemble prévue de valeurs futures.

La valeur présente et les valeurs passées ne peuvent pas changer.

Les valeurs futures sont reconsidérées à chaque fois qu'un élément le demande.

signal VALIDATION : BIT;



Classes d'objets (3)

Un signal est déclaré dans le domaine concurrent, ainsi il est accessible dans le domaine séquentiel. Une déclaration de signal peut être locale (voir blocs), néanmoins ce signal existe pendant toute la durée de la simulation.

Au temps t=0, tous les signaux sont créés, et le réseau de signaux ne changera pas jusqu'à la fin de la simulation.

Une mise à jour de signal peut entrainer des changements dans tout le modèle : l'algorithme "event-driven" est basé sur la sémantique de la propagation par les signaux.

Valeurs par défaut

Tous les signaux et variables peuvent se voir affecter une valeur initiale.

Tous les paramètres formels peuvent se voir affecter une valeur par défaut : Cette valeur sera utilisée si le paramètre manque (sous-programme), ou si le port n'est pas connecté (composants).

Tous les types ont une valeur par défaut : aucune valeur n'est inconnue au démarrage de la simulation (T=0).

Notation par Agrégats

En VHDL (comme ADA), il est possible d'écrire les constantes par regroupement de n'importe quel type.

```
Ex :  type TAB is array(1 to 5) of INTEGER;
      type ART is record      champ1 : INTEGER;
                              champ2 : BIT;
                              champ3 : INTEGER;
                end record;

 Affectation positionnelle :      VTAB := (5,6,7,8,9);      VART := (6,'0',8);

Affectation par dénomination : VTAB := (1=>5,2=>6,3=>7,4=>8,5=>9);
                               VART := (champ1=>6,champ2=>'0',champ3=>8);

Affectation mixte :                    VTAB := (5, others =>0);
                                       VART := champ2=>'1'; others =>0);
```

Sous-programmes

Il y a deux types de sous-programmes en VHDL :

Les **procédures**, agissant par l'intermédiaire de leurs arguments (**in**, **out** ou **inout**), et par effet de bord.

Les **fonctions**, retournant une valeur calculée d'après leurs arguments avec aucun effet de bord permis.

Les **fonctions de résolution** pouvant être appelées explicitement (dans une expression) ou implicitement (chaque fois qu'une source concernée doit être résolue).

Procédures

Déclaration (Package, ou recursivité)

```
procedure FOO(ARG : in TYPE; R : out INTEGER);
```

Corps

```
procedure FOO(ARG : in TYPE; R : out INTEGER) is
begin
.....
end FOO;
```

Utilisation : Domaine séquentiel FOO(Value,Int);

Domaine concurrent LABEL: FOO(signal_x,Int);

Fonctions

Déclaration (Package, ou recursivité)
function FOO(ARG : **in** INTEGER) **return** SOME_TYPE;

Corps
function FOO(ARG : **in** INTEGER) **return** SOME_TYPE **is**
begin
..... **return** some_value;
end FOO;

Utilisation : Domaine séquentiel Var := FOO(2);

Domaine concurrent Sign_x <= FOO(Signal_y);

Surcharge (1)

Les procédures, fonctions et littéraux énumérés peuvent partager le même nom, du moment que le contexte permet la différenciation :

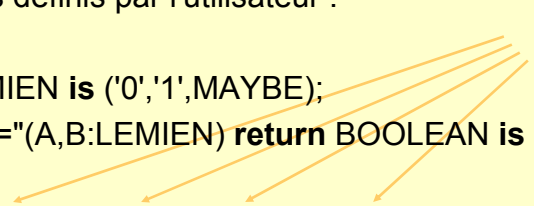
type COULEUR **is** (BLEU, ORANGE, ROUGE); --- 1
function ROUGE **return** INTEGER; --- 2
function ROUGE(C: CHARACTER) **return** INTEGER; --- 3
procedure ROUGE; --- 4
variable C:COULEUR; I:INTEGER;

C:= ROUGE; --- 1
I:=ROUGE; --- 2
I:= ROUGE('A') + ROUGE; --- 3 suivi par 2
ROUGE; --- 4

Surcharge (2)

Pas de défaut en cas d'ambiguité.

Les opérateurs prédéfinis("+","&", and ...) sont déjà surchargés dans le package STANDARD, et peuvent être surchargés avec les types définis par l'utilisateur :




```
type LE_MIEN is ('0','1',MAYBE);
function "="(A,B:LEMIEN) return BOOLEAN is
begin
return (A='0' and B='0') or (A='1' and B='1');
-- retourne FALSE si A=B=MAYBE, ce qui n'est pas le cas avec la
-- fonction "=" par défaut
end;
```


Surcharge (3)

```
type TPE is ....
subtype TPE2 is TPE1 range .... ;
type ENUM is (....,PP,....);

procedure PP (ARG : TPE1);
procedure PP (ARG : TPE2);
function PP (ARG : TPE1) return INTEGER;
function PP (ARG : TPE1) return CHARACTER;
procedure PP (ARG1,ARG2 : TPE1);
procedure PP(ARG1:TPE1;ARG2:TPE1:=default_value);
```

Surcharge (4)

```
type ENUM is (UN,DEUX,TROIS);
function UN return ENUM is
begin       return UN;
end UN;
type ENUM1 is (TROIS,DEUX,UN, ZERO);
function UN return ENUM1 is    begin return UN;
                               end UN;

type PT1 is access QUELQUE;
function NULL is               begin
                               return null;
end NULL;
type PT2 is access QUELQUE_CHOSE;
function NULL return PT2 is            begin
                                       return null;
end NULL;
```


Le domaine séquentiel (1)

Les "**process**" et **sous-programmes** sont construits avec des instructions séquentielles.

L'ordre des instructions est important, comme dans tout langage de programmation.

Les "process" et sous-programmes peuvent comprendre des instructions d'attente ("**wait**"), permettant le lien avec le domaine concurrent.

Le domaine séquentiel (2)

```
A := B;
B :=C;
```

```
B := C;
A :=B;
```

Ces deux séquences ne sont pas équivalentes

Le domaine concurrent (1)

Vu de l'extérieur, une description VHDL est un réseau statique de composants statiques connectés entre eux.

La description est une suite d'instructions concurrentes.

Le résultat est indépendant de l'ordre d'évaluation par le simulateur.

Une des instructions concurrentes est le **process**, d'autres permettent d'appeler des **sous-programmes**.

Le domaine concurrent (2)

```
A <= B;
B <=C;
```

```
B <= C;
A <=B;
```

Ces deux séquences sont équivalentes
Lien permanent entre A,B,C
Si C change, B change, puis A

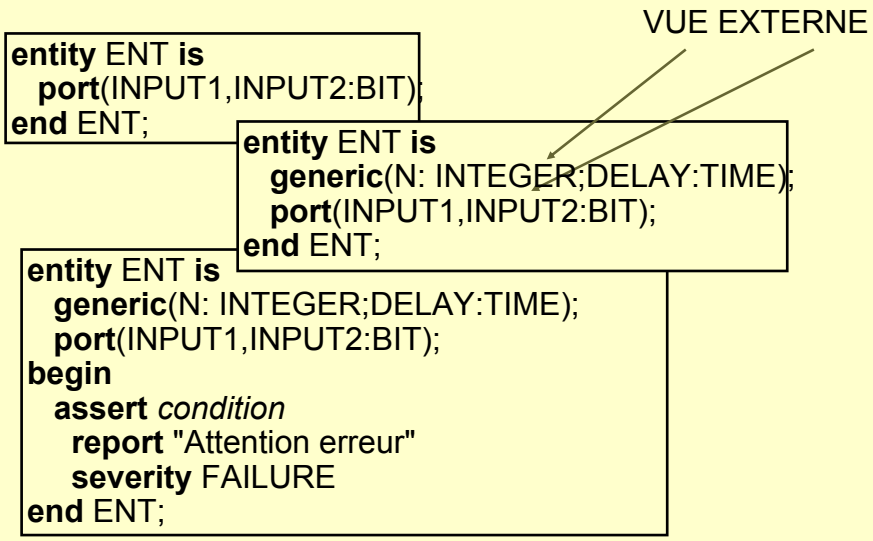
Le Package

Utilisable par Unité(s) de conception avant création du corps

```
package PACK is
-- Déclarations de types, sous-types, signaux, composants
-- Constantes, sous-programmes (sans code). Aucune Variable
end PACK;
```

```
package body PACK is
-- Déclarations identiques (sauf signaux) + corps des
-- sous-programmes de la partie déclarative
end PACK;
```

Déclaration d'Entité



Architecture

clauses *library & use*

architecture ARCH **of** ENT **is**

-- Déclarations de types, sous-types, constantes, signaux

-- sous-programmes, composants .. PAS de VARIABLES

begin

-- Liste d'instructions concurrentes

-- *block* et *process* sont des instructions concurrentes

end ARCH;

Instruction Block

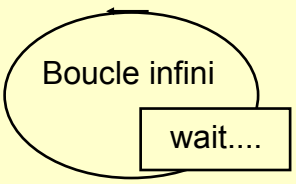
Label_1 : **block** *(Guard_expression)*
generic (...)
generic map (...)
port (...)
port map (...)
partie_declarative
begin
partie_instructions_concurrentes
end block Label_1;

Signaux gardés

Le process (Instruction concurrente)

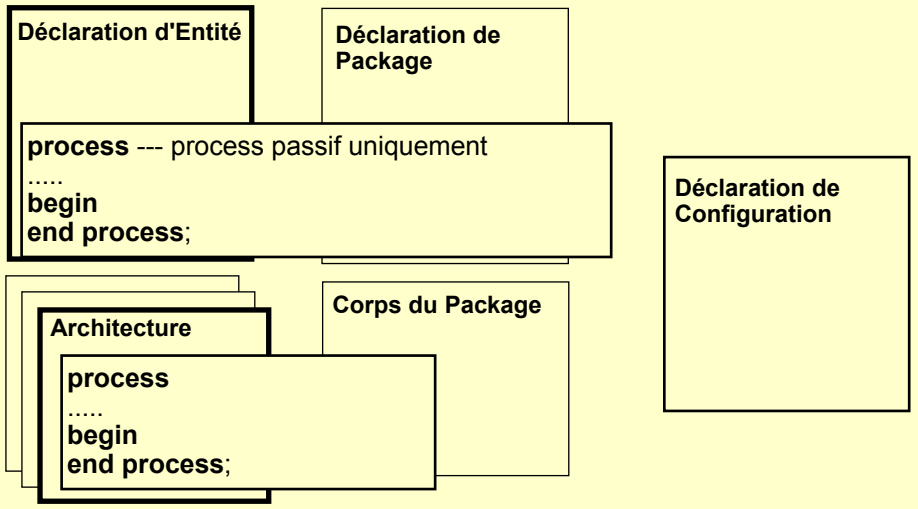
Label_P1 : **process**
variable V: TPE; --- permis ici , PAS de SIGNAUX
begin

V existe toute la simulation
Mise à jour par le process



end process Label_P1;

Le process (où)



Le process (Visibilité des signaux)

signal S1,S2 : TYPE_X;

visible du process

```
process
 variable V: TYPE_UN;
begin
  V:= ..... -- par exemple V := S1
  S1 <= ...  -- par exemple S1 <= V
  wait on S2;
-- ou wait until condition, ou wait for 5 ns;
end process;
```

Le process (Liste de sensibilité)

Process équivalents

```
process (S1,S2)
 variable V: TYPE_UN;
begin
  V:= ........
  S1 <= ...
end process;
```

```
process
 variable V: TYPE_UN;
begin
  V:= ........
  S1 <= ...
  wait on S1, S2;
end process;
```


Le Signal (1)

Déclaration :

signal S : Type_du_signal; --- tout sauf access ou file

Utilisation :

Domaine concurrent : S <= S1 after 5 ns;

S <= FONC(S1) **after** 5 ns, '1' **after** 15 ns;

Domaine séquentiel : Une_variable := S;

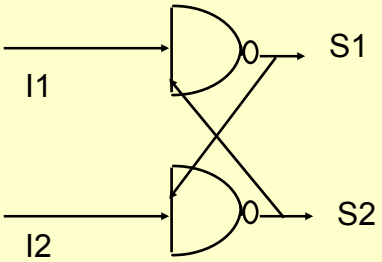
S <= Une_variable;

S <= Un_autre_signal;

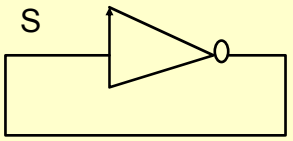
Le Signal (2)

S1 <= **not** (I1 **and** S2);

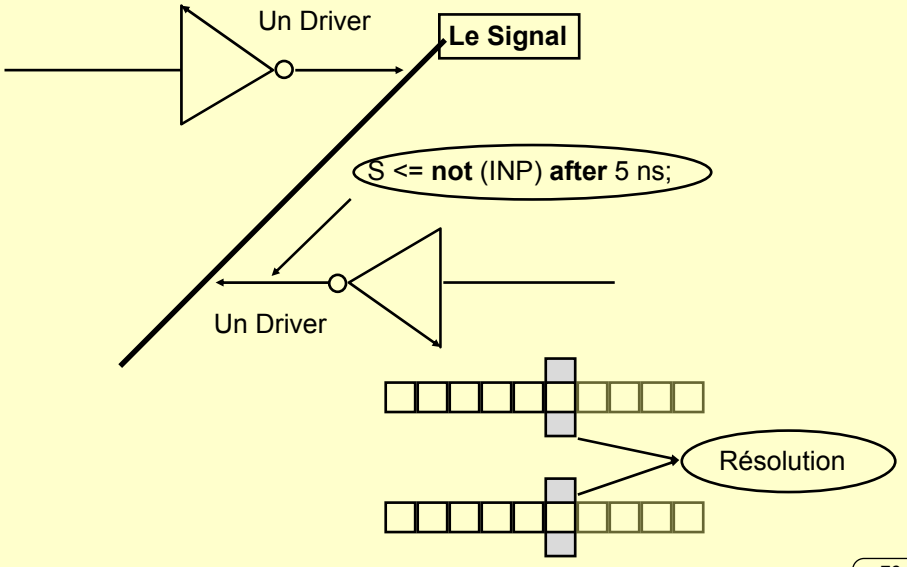
S2 <= **not** (I2 **and** S1);



S <= **not** S ;



Signaux et Pilotes (Drivers)



Processus / Signaux / Pilotes

DOMAINE CONCURRENT

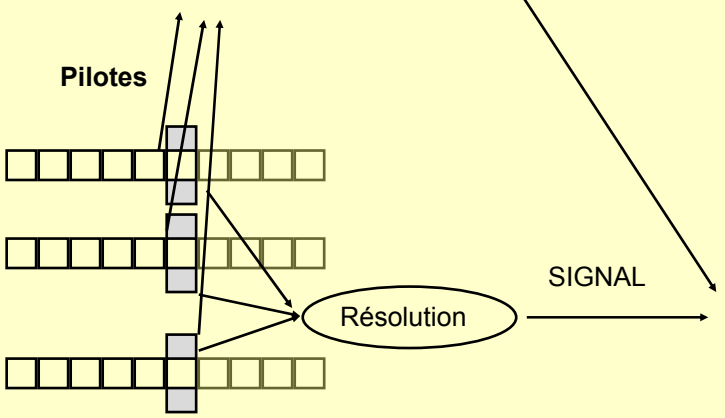
```
process   begin
          S <= valeur after Time;

          if ..... then S<= Autre_Valeur after Time;
          S1 <= .....;
end process;
```

```
S <= ENABLE after 10 ns;
```

Un pilote par signal par processus

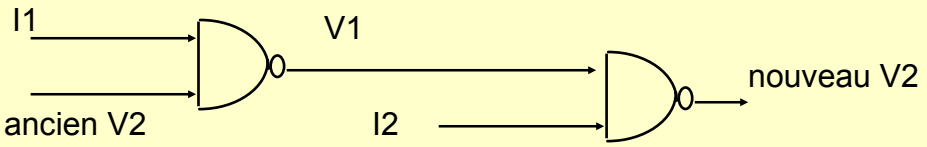
Pilotes et Résolution

```
function RESOLVE (ARGS : in BIT_VECTOR) return BIT;
```

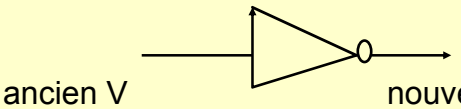


Différences Signal / Variable

V1 := **not** (I1 **and** V2); -- ancien V2
V2 := **not** (I2 **and** V1) ; -- nouveau V2



V := **not** V;



Instructions VHDL

Instructions séquentielles

Instructions concurrentes

Instructions séquentielles

POSITION :

Processus

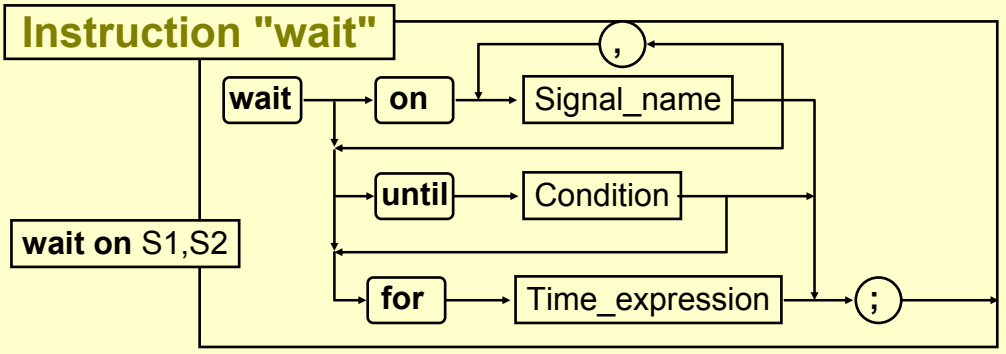
Sous-programmes

FORMAT :

Simple : wait, assert, affectation, appel de procédure
return, exit, next

Composé : if, case, loop

Instruction "wait"



wait on S1,S2

wait on CLK **until** D='1';

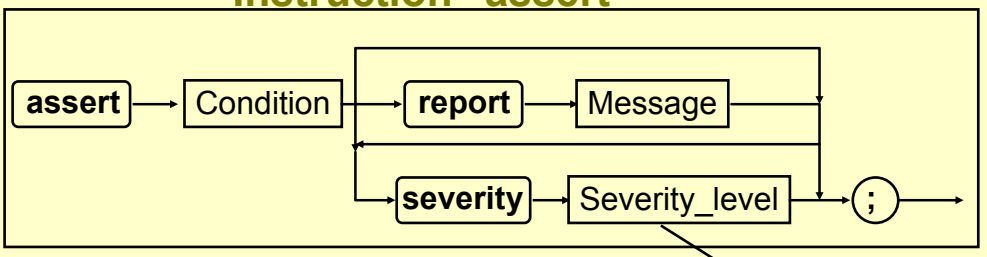
wait until D='1'

```
loop
wait on CLK;
exit when D='1';
end loop;
```

wait on S1 **until** D='1' **for** 10 ns;

wait until ((CLK'event) **and** (CLK='1') **and** NOT (CLK'stable))

Instruction "assert"



Note
Warning
Error
Failure

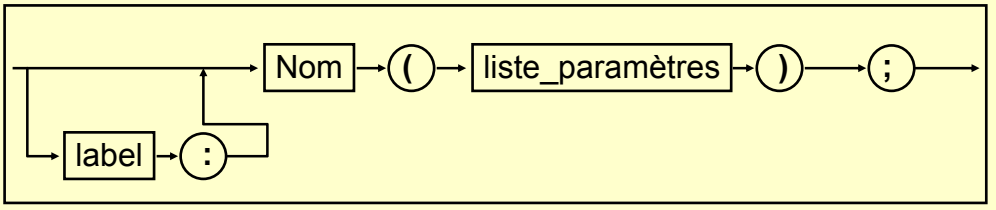
assert *une_condition* **report** "Un message" **severity** Warning;

assert *une_condition* **report** "Un message" **;**

assert *une_condition* **severity** Error **;**

assert *une_condition* **;**

Instruction d'appel de procédure

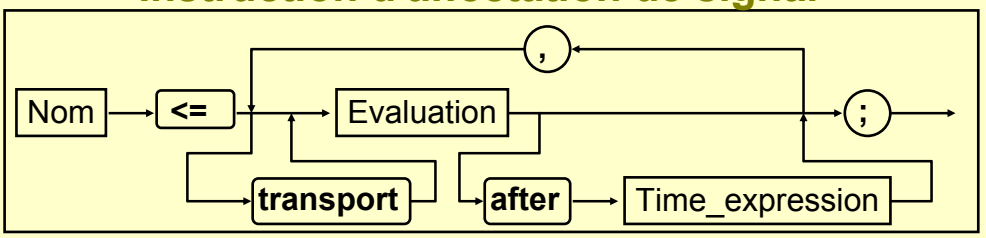


PROCEDURE (1, Une_variable, Un_signal);

PROCEDURE;

PROCEDURE (Formal1 => 3, Formal2 => "Hello");

Instruction d'affectation de signal



A <= B;

A <= **transport** B **after** 5 ns;

A <= **transport** B **after** 10 ns;

Data <= 2 **after** 1 ns, 4 **after** 3 ns, 10 **after** 8 ns;

A <= ma_fonction(toto,4) **after** 7 ns;

A <= B **after** 10 ns;

Bus1 <= B"01" **after** 30 ns;

Affectation de variable

VAR := 0;

A := B;

Str1 := "Mon Nom est " & Intro_nom & " , et le vôtre ? ";

A := Ma_fonction(data,6);

Variant := Fonc(data,8) + 243 * Alter(variant) - 2 ** data;

Instruction de retour de valeur (fonction)

```
return;
```

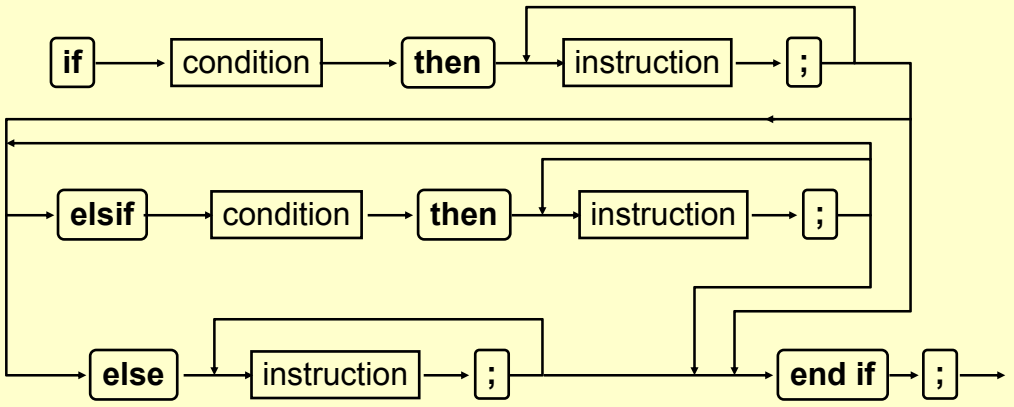
```
return valeur;
```

```
return A+B+ 10 ns;
```

```
return fonction(data, 5 pF);
```

```
return "Nom de l'Auteur : " & Name;
```


Instruction "if...then....else"



Instruction "if...then...else"

```
if A='1' then
        B := 0;
end if;
```

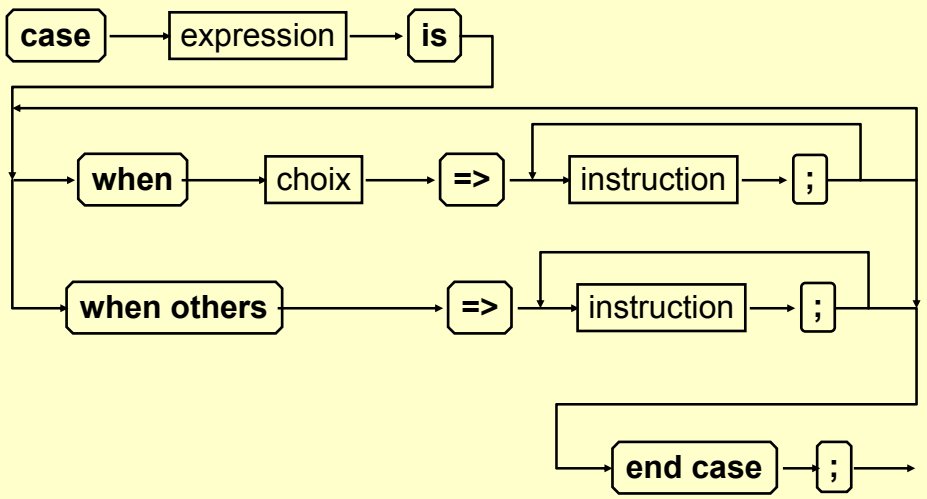
```
if clock'event and clock='1' then
        Q<=D after 5 ns;
end if;
```

```
if val>=5 and val<10 then
 int:=7;
elsif val <5 then int:=val + func(val+2);
elsif val<15 then int:=func(val);
else int:=0;
end if;
```

```
if A > 0 then
        B:=A;
else
        B:=abs(A+1)
end if;
```

```
if reset='0' then
  Q<='0' after 1 ns;
elsif clock'event and clock='1' then Q<=D after 5 ns;
end if;
```


Instruction "case"



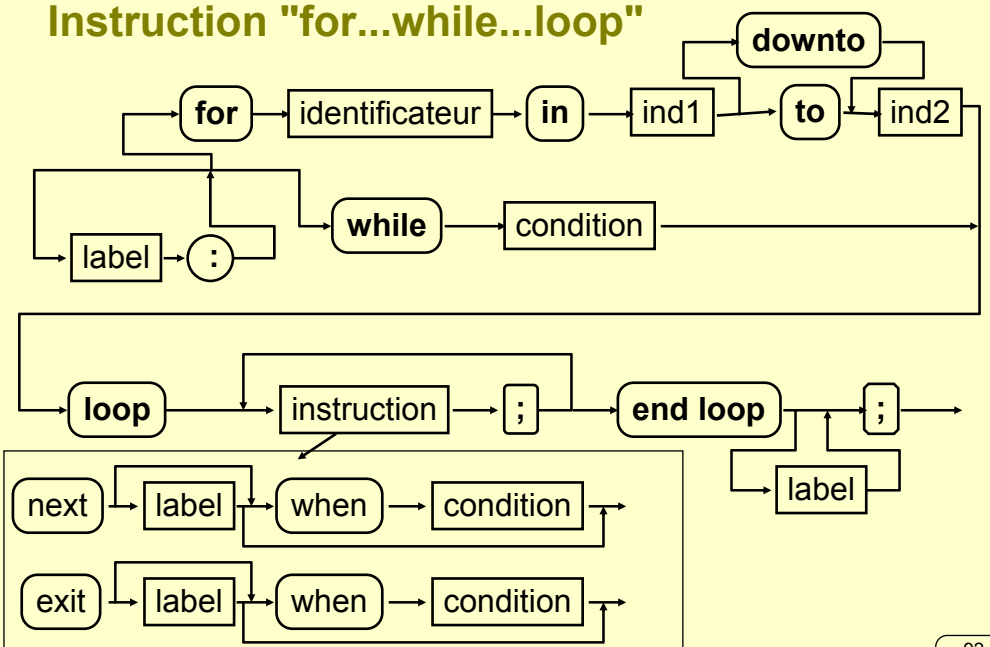
Instruction "case"

```
case A is
        when '0' => S <= "000" after 2 ns;
        when '1' => S <= "111" after 2 ns;
end case;
```

```
case two_bit'(A_en & B_en) is
        when "00" => S <= zero after 1 ns;
        when "01" => S <= A after 1 ns;
        when "10" => S <= B after 1 ns;
        when "11" => S <= one after 1 ns;
end case;
```

```
case val_int is
        when 0          => int:=5;
        when 1|2|8      => int:=val_int;
        when 3 to 7     => int:=val_int + 5;
        when 9          => null;
        when others     => int:=0;
end case;
```


Instruction "for...while...loop"



Instruction "for...while...loop"

```
loop
wait until clock='1';
Q<=D after 1 ns;
end loop;
```

```
lab1: for i in 0 to 5 loop
        var := var + 3;
end loop;
```

```
while val > 0 loop
        next when val=3;
        tab(val) := val rem 2;
        val := val / 2;
end loop;
```

```
for i in 10 downto  2 loop
        for j in 0 to i loop
        table(i,j) := i + j + 3;
        end loop;
end loop;
```

```
L1: for i in 10 downto 0 loop
  L2: for j in 0 to i loop
        exit L2 when i = j;
        table(i,j) := i + j + 3;
  end loopL2;
end loopL1;
```


Instructions concurrentes

POSITION :

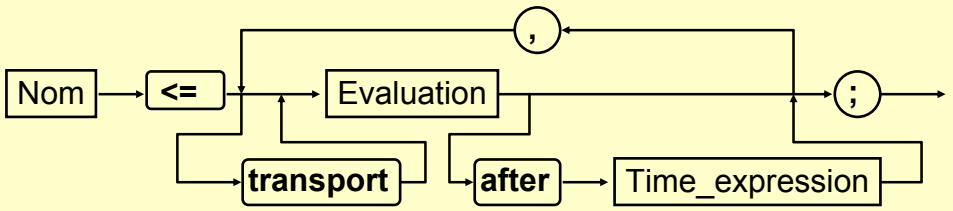
Déclaration d'entité
Corps de l'architecture
Instruction "block"

FORMAT :

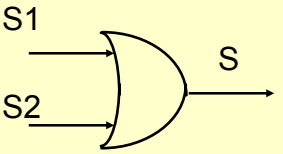
Actives au "run-time" : Affectation de signal
Appel concurrent de procédures
Instruction "process"
Instruction "assert"

Actives à l'élaboration : Instanciation
Instruction "generate"
Instruction "block"

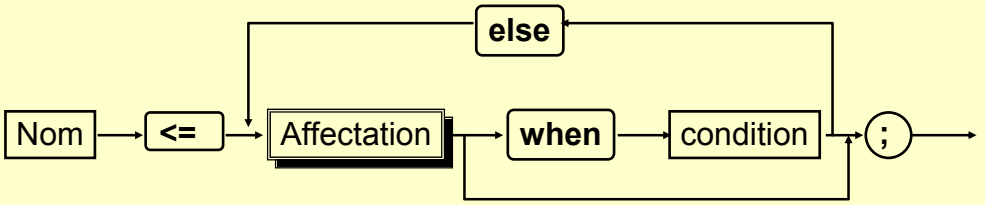
Affectation de signal



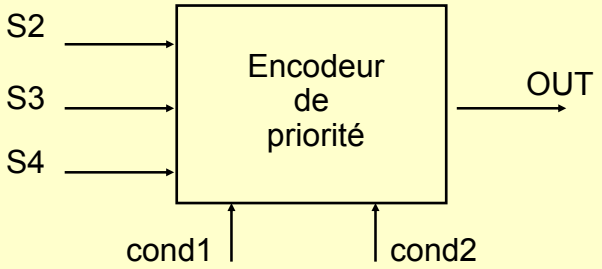
S1 <= S2 **or** S3 **after** 5 ns;



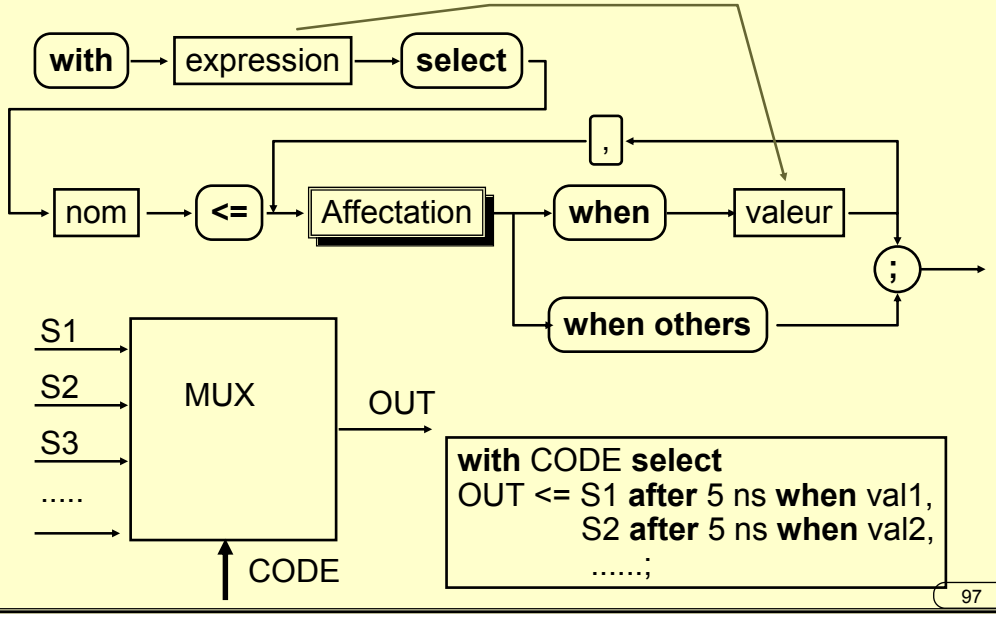
Affectation conditionnelle



OUT <= S2 **when** cond1 **else** S3 **when** cond2 **else** S4;



Affectation sélective




```
with CODE select
OUT <= S1 after 5 ns when val1,
       S2 after 5 ns when val2,
       ......;
```


Le process équivalent (1)

```
S1 <= S2 or S3 after 5 ns;
```

```
process
begin
  S1 <= S2 or S3 after 5 ns;
  wait on S2, S3;
end process;
```

Le process équivalent (2)

S1 **<=** S2 **when** Cond **else** S3 ;

```
process
begin
  if Cond then S1 <= S2;
  else S1 <= S3;
  end if;
  wait on S2, S3;
end process;
```

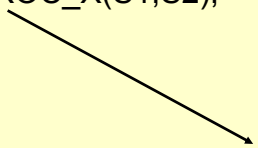

Le process équivalent (3)

```
with CODE select
S1 <= S2 after 5 ns when val1,
        S3 after 4 ns when val2,
         .....;
```

```
process
begin
  case CODE is
  when Val1 => S1 <= S2 after 5 ns;
  when Val2 => S1 <= S3 after 4 ns;
 ...........;
  wait on S2, S3;
end process;
```

Le process équivalent (4)

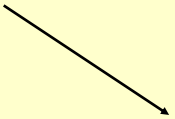
LBL : PROC_X(S1,S2);



```
LBL : process
begin
  PROC_X (S1,S2);
  wait on S1, S2;
end process LBL;
```


Le process équivalent (5)

```
assert (S1 /= '0' ) and (S2 /= '0')
        report "Message pour l'opérateur"
        severity warning;
```

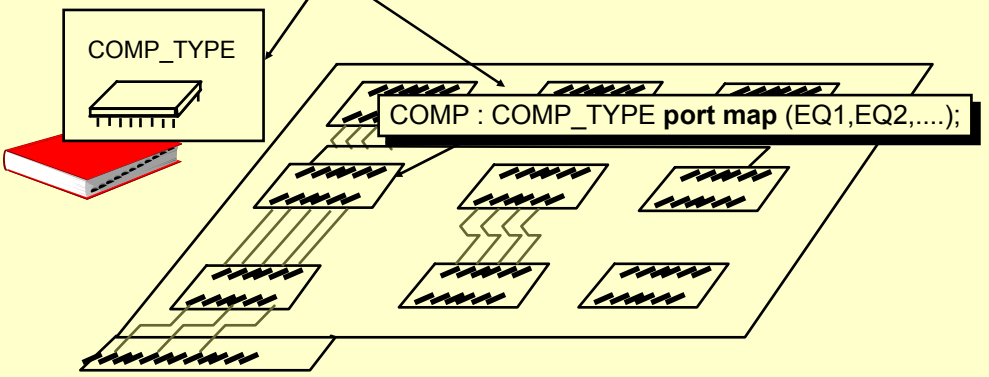


```
process
begin
assert (S1 /= '0' ) and (S2 /= '0')
        report "Message pour l'opérateur"
        severity warning;
wait on S1, S2;
end process;
```

La liste d'équipotentielles (Netlist)

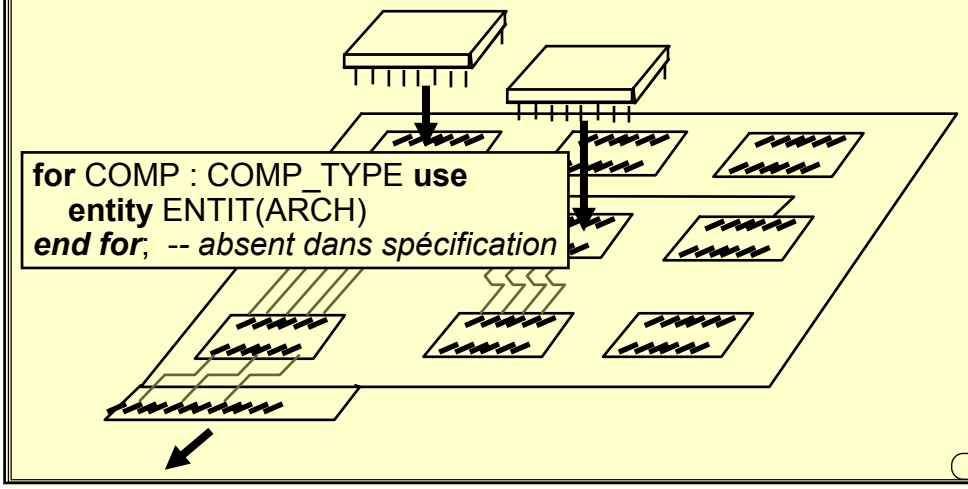
component COMP_TYPE
port(I1,I2 : in BIT; ...)
end component;

Le niveau structurel décrit uniquement une "netlist"
La liaison avec l'entité réelle n'est pas dans la description structurelle



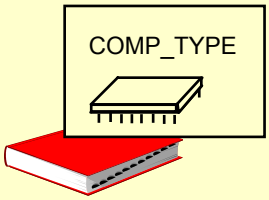
La Configuration

La configuration fait le lien entre les instances et les entités (couple entité/architecture)
Cette configuration peut être utilisée dans une autre configuration



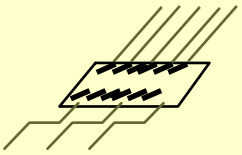
Le composant

component COMP_TYPE
port(I1,I2 : in BIT; ...)
end component;



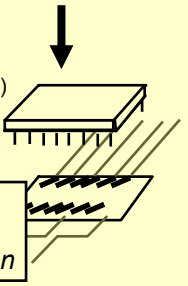
Déclaration

Instanciation



COMP : COMP_TYPE **port map** (EQ1,EQ2,....);

Configuration



for COMP : COMP_TYPE **use**
entity ENTIT(ARCH)
end for; -- *absent dans spécification*

Déclaration de composant (où)

Déclaration d'Entité

Déclaration de Package

component COMP_TYPE **port**(...) **end component**;

Déclaration de Configuration

Corps du Package

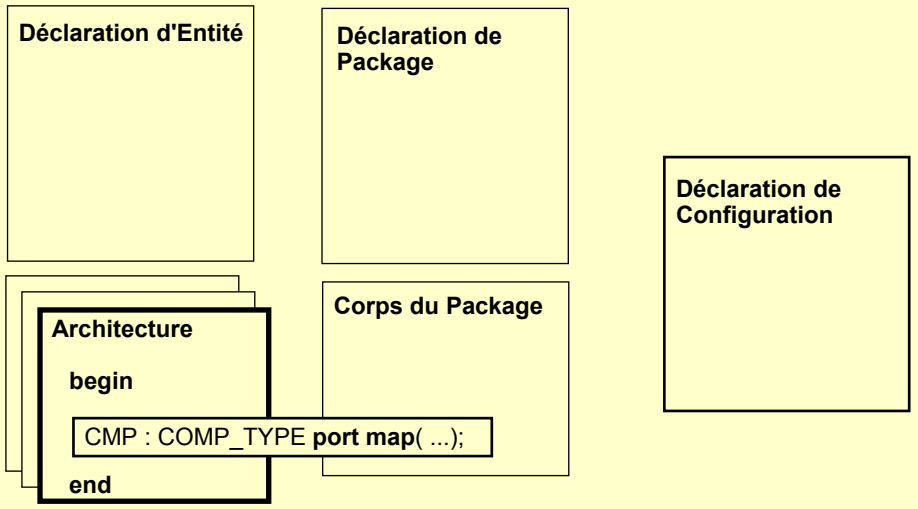
Architecture

component COMP_TYPE **port**(...) **end component**;

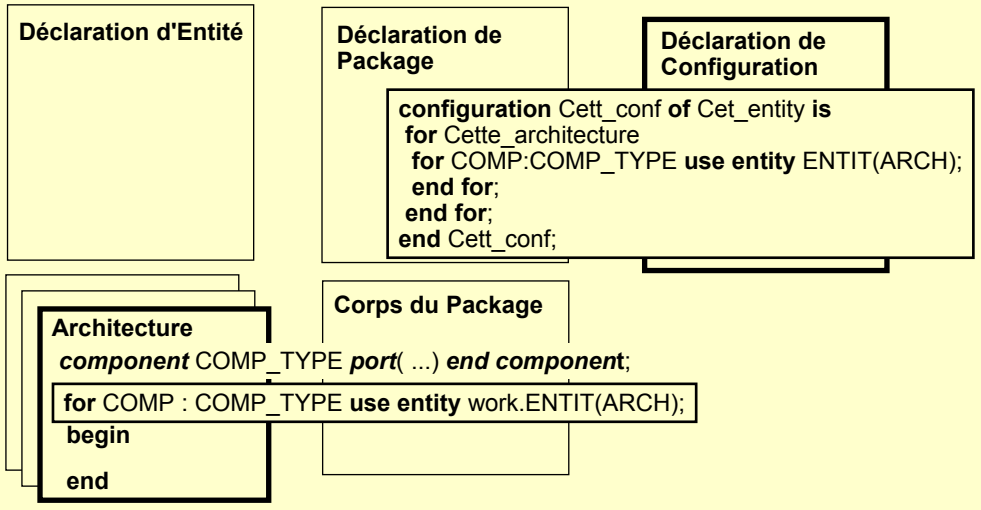
begin

end

Instanciation de composant (où)



Configuration de composant (où)



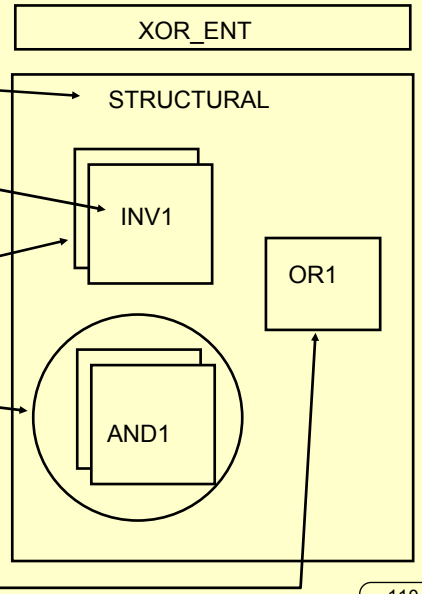
Spécification de configuration

```
for INV1 : INVCMP use
 entity work.INVENT(DATAFLOW);
```

```
architecture STRUCTURAL of XOR_ENT is
 component INVCMP
    port (Input: in BIT; Output : out BIT);
 end component;
 component ANDCMP
    port (Input1, Input2 : in BIT; Output :out BIT);
 end component;
 component ORCMP
    port (Input1, Input2 : in BIT; Output :out BIT);
 end component;
signal S1, S2, S3, S4 : BIT;
begin
    INV1 : INVCMP port map (Input1, S1);
    INV2 : INVCMP port map (Input2, S2);
    AND1 : ANDCMP port map (S1, Input2, S3);
    AND2 : ANDCMP port map (S2, Input1, S4);
    OR1 : ORCMP port map (S3, S4, Output);
end STRUCTURAL;
```

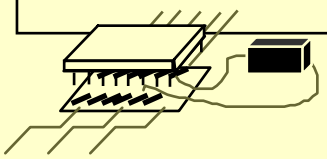

Déclaration de configuration

```
configuration MOI of XOR_ENT is
 for STRUCTURAL

  for INV1:INVCMP use
    entity INVENT(DATAFLOW)
  end for;

 for others : INVCMP use .....

 for all : ANDCMP use ......

 for OR1 : ORCMP use .....
end for;

end MOI
```



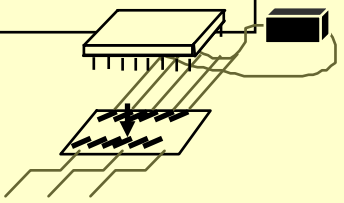
Adaptations

```
COMP : COMP_TYPE port map (BUS0 => DATA1, BUS1 => DATA0,
                           TO_BIT(COND) => TO_BOOLEAN(TST),
                           ENABLE => open);
```



L'instance de composant, ou la déclaration peut échanger des broches, en laisser ouverte, ou convertir les types.

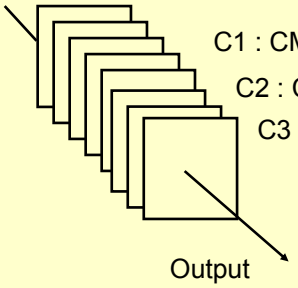
```
for COMP : COMP_TYPE use ENTIT(ARCH)
                    port map (BUS0 => DATA1, BUS1 => DATA0,
                              TO_BIT(COND) => TO_BOOLEAN(TST),
                              ENABLE => open);
end for;
```



Génération (1)

```
port (Input : in BIT; Output  out BIT);

signal S2,S3,S4,S5,S6,S7,S8 : BIT;
```

Input



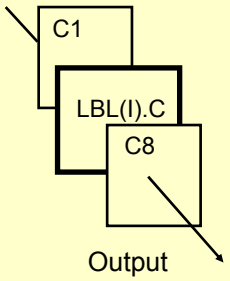
Output

```
C1 : CMP port map (....., Input,S2,.....);
C2 : CMP port map (.....,S2,S3,.....);
C3 : CMP port map (....., S3,S4,.....);
C4 : CMP port map (....., S4,S5,.....);
C5 : CMP port map (....., S5,S6,.....);
C6 : CMP port map (....., S6,S7,.....);
C7 : CMP port map (....., S7,S8,.....);
C8 : CMP port map (....., S8,Output,.....);
```

Génération (2)

port (Input : **in** BIT; Output **out** BIT);

Input



signal S : BIT_VECTOR(2 to 8);

C1 : CMP **port map** (....., Input,S(2),.....);

C8 : CMP **port map** (....., S(8), Output.....);

LBL : **for** I in 2 to 7 **generate**
C : CMP **port map** (....., S(I), S(I+1).....);
end generate;

Paramètres génériques (1)

```
entity BROF is
 generic (X: INTEGER; T: TIME)
end BROF;

architecture BEHV of BROF is
   S: BIT_VECTOR (1 to X);
begin
....
    for I in X downto 1 loop
.....
S(X) <= ..... after T;
.....
end BEHV;
```

Paramètres génériques (2)

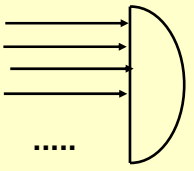
```
entity AND4 is
 port(I1,I2,I3,I4 : in BIT; O : out BIT);
end AND4;
                                DATF of AND4 is
```

```
entity AND2 is
 port(I1,I2 : in BIT; O : out
   BIT);
end AND2;
architecture DATF of A

begin
O <= I1 and I2;
end DATF;
```

```
entity AND3 is
 port(I1,I2,I3 : in BIT; O : out BIT);
end AND3;
architecture DATF of AND3 is

begin
O <= I1 and I2 and I3;
end DATF;
```


Paramètres génériques (3)

```
entity ANDGEN is
   generic (N: in POSITIVE); -- subtype 1 à max_integer
   port (INPUTS  in BIT_VECTOR(1 to N); O : out BIT);
end ANDGEN;
architecture DATF of ANDGEN is
begin
   process
 variable Result : BIT := '1';
        begin
                for I in INPUTS'RANGE loop
                RESULT := RESULT and INPUTS(I);
                end loop;
        O <= RESULT;
   wait on INPUTS;
   end process;
end DATF;
```



Paramètres génériques (4)

```
for L1,L2 : PORTE_AND3 use entity work.ANDGEN(DATF)
        generic map (N =>3);
        port map (INPUTS(1) => S1, .....);

for all : PORTE_AND2 use entity work.ANDGEN(DATF)
        generic map (N=>2);
        port map (INPUTS(1) => S1, ...);
```


Paramètres génériques & génération

```
entity REG_ENT is
  generic (N : in POSITIVE);
  port(....);
end REG_ENT;

architecture STRUCT of REG_ENT is
  use UNE_BIBLIO.UN_PACKAGE.all;
begin
 LAB1: for I in 1 to N generate
    LAB2 : if I = 1 generate              C1: SLICE port map (.....);
           end generate;
    LAB3 : if I > 1 and I < N generate    CN: SLICE port map (....);
           end generate;
    LAB4 : if I = N generate              C8 : SLICE port map (.....);
           end generate;
 end generate;
end STRUCT;
```

Différence entre "generic" et constantes

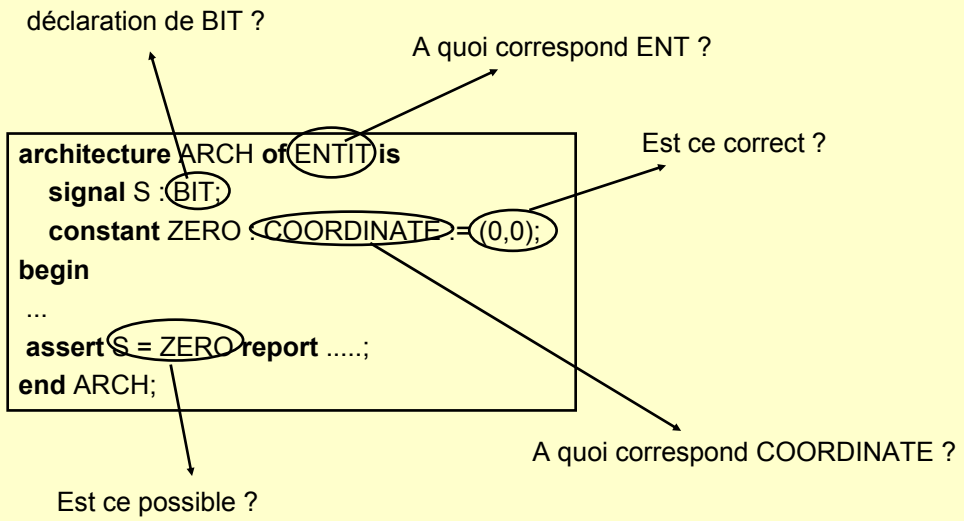
```
entity ENTIT is
  generic (Delay : in TIME);
  port (.....);
end ENTIT;

architecture STRUCTURAL of ENTIT is

begin
.....
end STRUCTURAL;
```

Les # instances peuvent avoir des délais différents

```
entity ENTIT is
   port (.....);
   constant delay : TIME := work.monpack.csttime;
end ENTIT;

architecture STRUCTURAL of ENTIT is

begin
.....
end STRUCTURAL;
```

Les # instances ont toutes le même délai

Librairies (1)



Librairies (2)

Package STANDARD

```
type BIT is ('0','1');
type INTEGER is range ... to ...;
```

Déclaration d'entité

```
entity ENTIT is
...
end ENTIT;
```

Package utilisateur

```
type COORDINATE is record
  X,Y : INTEGER;
end record;
```

Contexte de compilation

```
use WORK.pack_util.all;
architecture ARCH of ENTIT is
   signal S : BIT;
   constant ZERO : COORDINATE := (0,0);
begin
 ...
 assert S = ZERO report .....;
end ARCH;
```

Affectation correcte pour le type

Incompatibilité de type entre S et ZERO

Librairies (3)

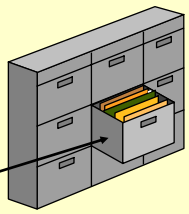
Index dépend du système

```
library LIB;
use LIB.unit_un.all;
 (explicite)
```

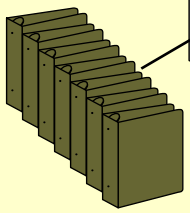
```
library STD;
use STD.STANDARD.all;
 (implicite)
```

Unité actuelle

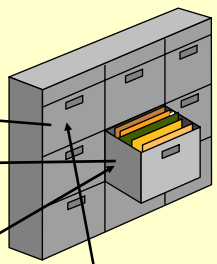
```
library WORK;
use WORK.REGIS.all;
```

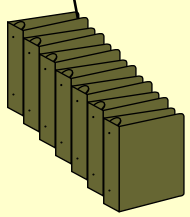


Library WORK;



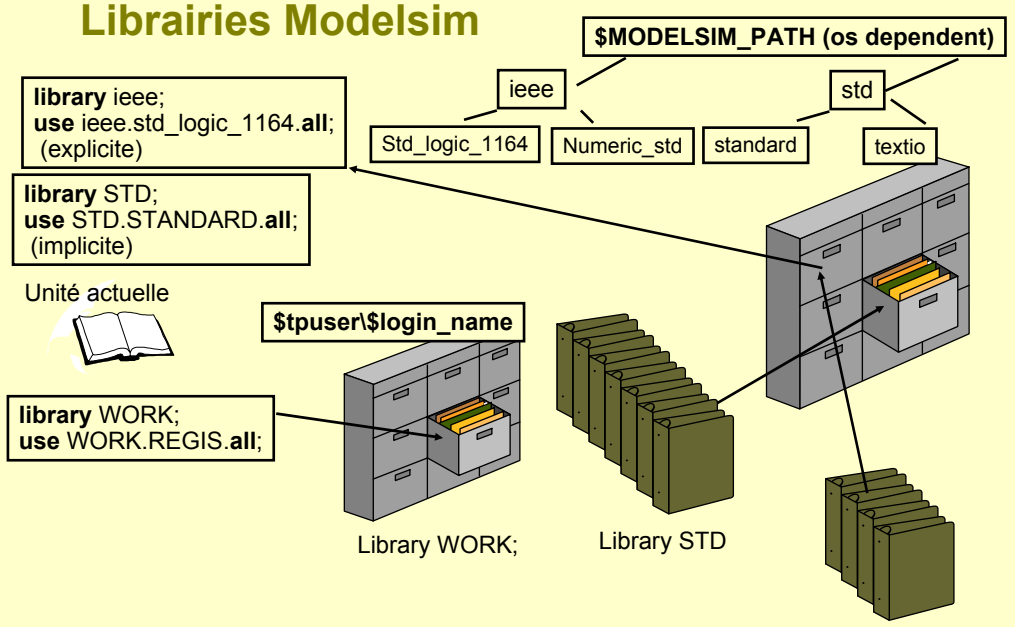
Library STD



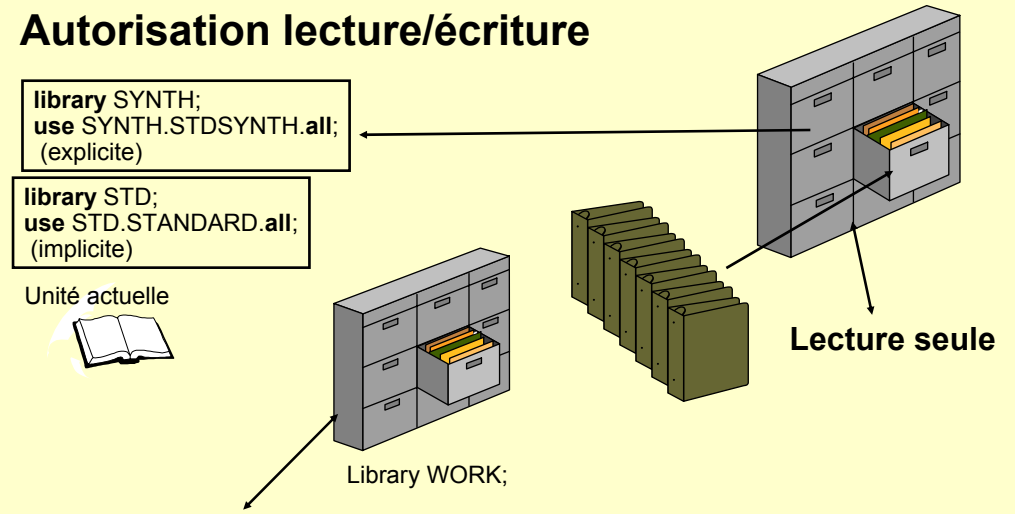


Library user_defined

Librairies Modelsim

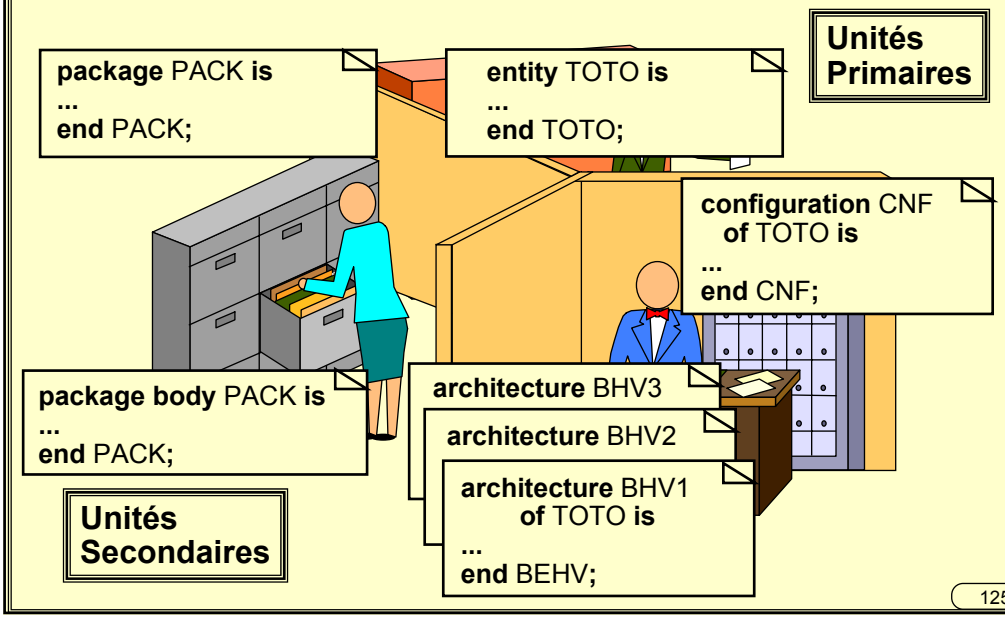


Autorisation lecture/écriture



Les protections (R/W) sont assurées par l'OS

Visibilité des unités



Ordre de compilation

1) Le Package STANDARD

```
type BIT is ('O','1');
type INTEGER is range ... to ...;
```

3 ou 2) Le Package utilisateur

```
type COORDINATE is record
  X,Y : INTEGER;
end record;
```

2 ou 3) La Déclaration d'entité

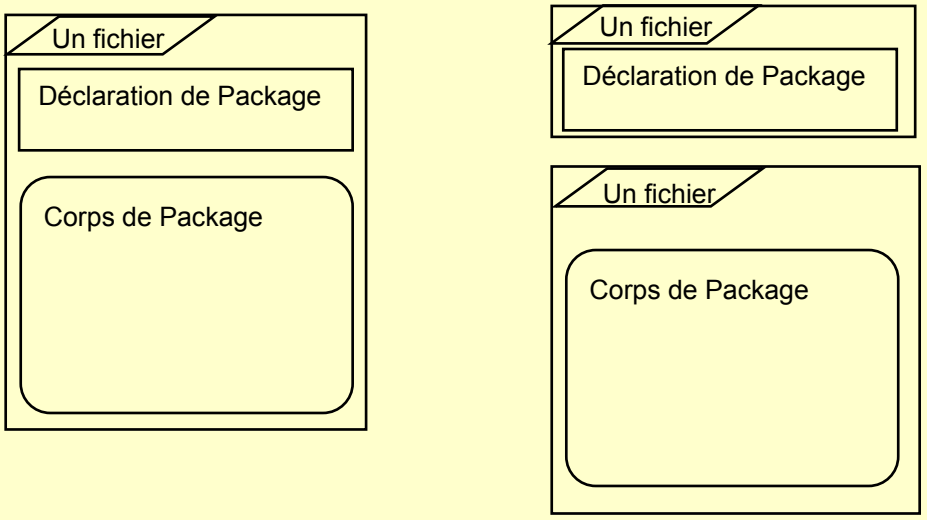
```
entity ENTIT is
...
end ENTIT;
```

4) L'unité de conception

```
use WORK.pack_util.all;
architecture ARCH of ENTIT is
   signal S : BIT;
   constant ZERO : COORDINATE := (0,0);
begin
 ...
 assert S = ZERO report .....;
end ARCH;
```

Le package utilisateur doit être compilé **avant** l'architecture.
L'entité ENTIT doit être compilé **avant** l'architecture.
Il n'y a pas de contrainte entre l'entité et le package.
Le package STANDARD doit être visible **avant** toute compilation

Unités de conception



Librairies et clauses "use"

library *STD;*
use *STD.STANDARD.**all**;* ***IMPLICITE***

```
library WORK;
use WORK.REGS.all;
architecture BEHV of ALU is
.....
  signal S: WORK.UN_PACK.UN_TYPE;
  use WORK.UN_PACK.UN_TYPE;
  signal S1 : UN_TYPE;
```

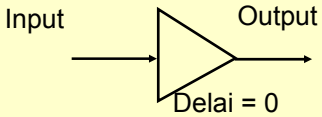
Calcul binaire ?

numeric_std.vhd (ieee) opérations sur unsigned et signed (complément à 2) de std_logic

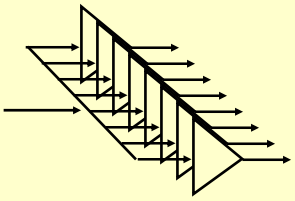
numeric_bit.vhd (ieee) opérations sur unsigned et signed de bit

Les anciens packages de type **std_logic_arith** (synopsis) ou **std_arith** (cypress) sont fonctionnels (certains sur std_logic d'ailleurs plutôt que « unsigned » ou « signed ») mais à utiliser uniquement avec parcimonie.

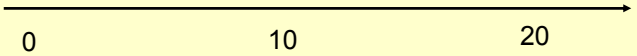
Le délai interne (simulateur)



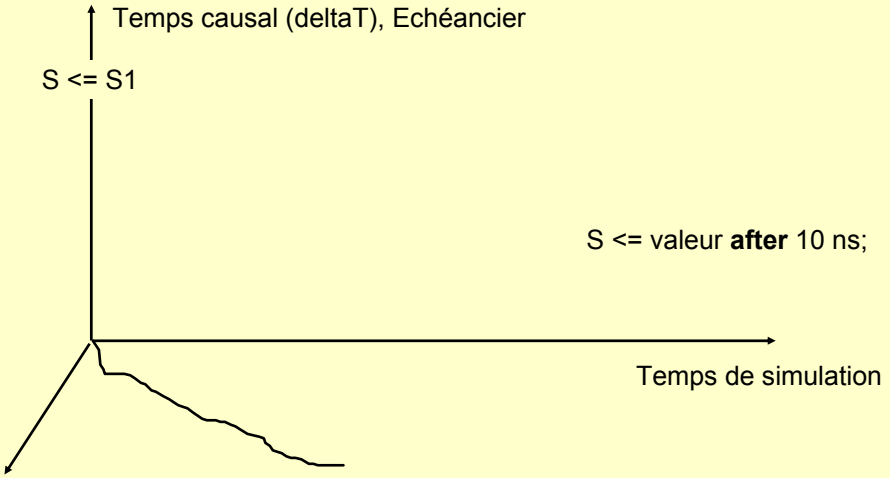
Chaque changement de la sortie est *causée* par un changement de l'entrée, même si cela arrive au même instant de simulation.
Causalité => 1 deltaT entre entrée et sortie



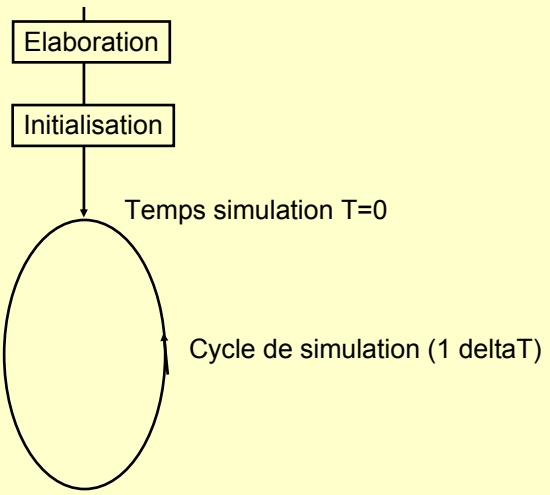
"Scheduler" (simulation)



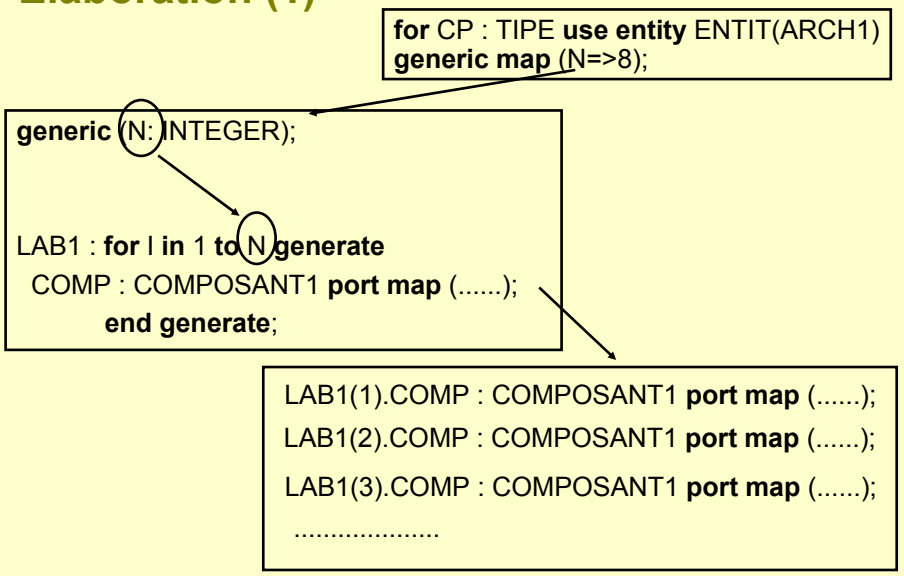
Les trois temps



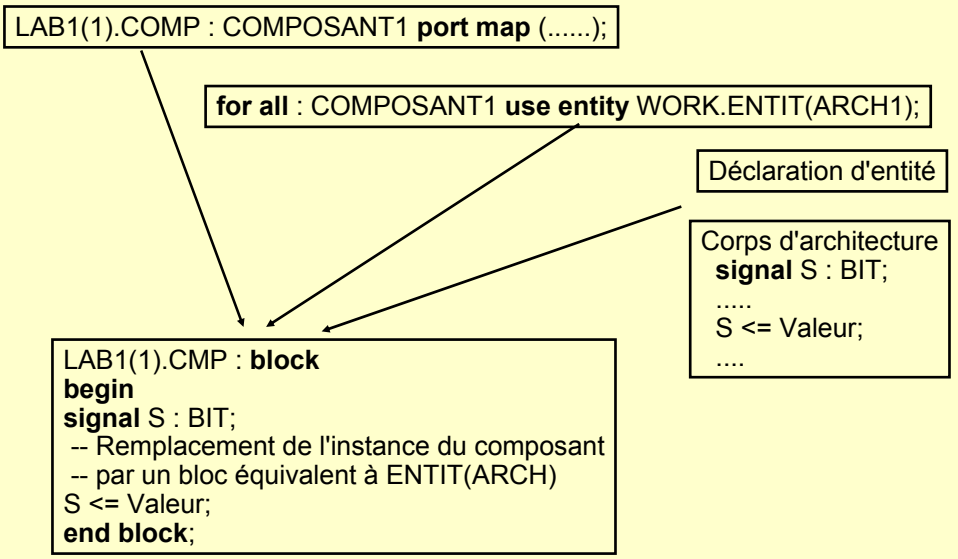
Elaboration - Initialisation - Simulation



Elaboration (1)



Elaboration (2)



Elaboration (3)

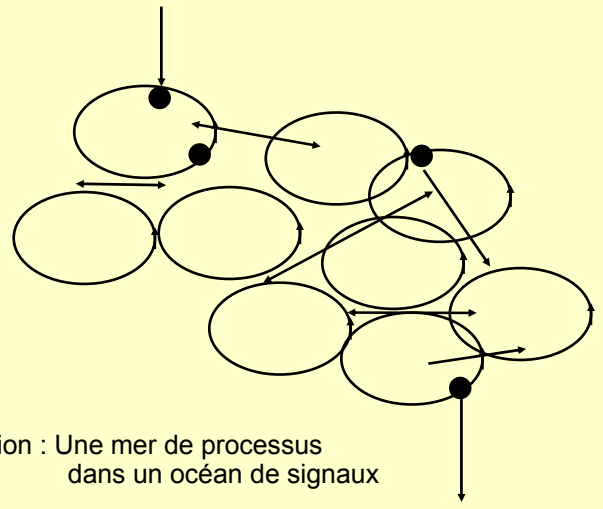
```
LAB1(1).CMP : block
begin
signal S : BIT;
 -- Remplacement de l'instance du composant
 -- par un bloc équivalent à ENTIT(ARCH)
S <= Valeur;
end block;
```

Création du signal S

```
process
begin
S <= Valeur;
wait on Signaux_impliqués;
end process;
```

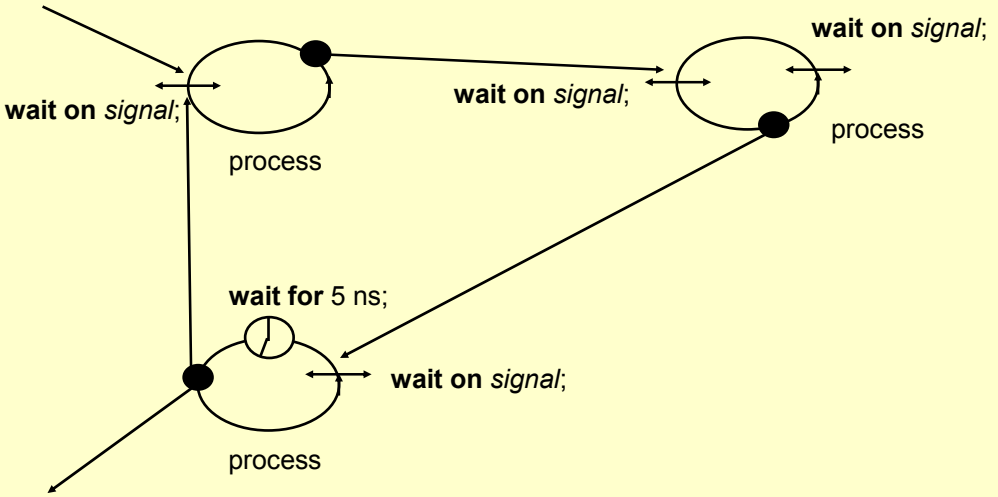
Création d'un pilote pour S

Les différents processus (Vue d'ensemble)



Simulation : Une mer de processus
dans un océan de signaux

Les différents processus (détail)



Initialisation

type ENUM **is** (ROUGE,ORANGE,VERT);

La valeur gauche est la valeur initiale par défaut

signal S ENUM; ---- := ROUGE;

variable V : ENUM := ORANGE;
-- initialisation explicite

```
process
begin
S <= Valeur;
wait on Signaux_impliqués;
end process;
```

Tous les process sont exécutés une fois **jusqu'à la première instruction wait**

Bloc gardé et affectation gardée

```
signal guard  : Boolean; ---- Déclaration implicite de signal

LAB1 : block (not CLOCK'STABLE and CLOCK = '1' )
          --- front montant
begin
Result <= guarded Arg1 + Arg2;

Autre_signal <= Valeur;
end block LAB1;
```

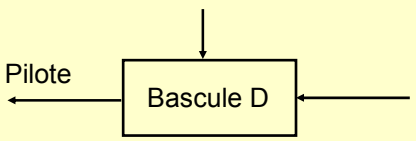
Affectation gardée

Affectation normale

Affectation gardée (process équivalent)

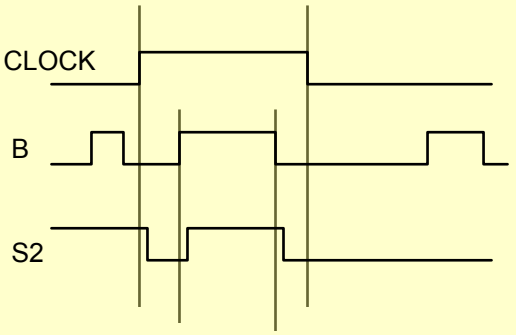
```
signal Result : [Fonc_res] BIT32; ---- Signal [résolu] ou non

LAB1 : block (not CLOCK'STABLE and CLOCK = '1' )
begin
guard <= not CLOCK'STABLE and CLOCK = '1'
--- Affectation implicite de signal
Result <= guarded Arg1 + Arg2;

Autre_signal <= Valeur;
end block LAB1;
```

```
LAB1 : process (CLOCK)
begin
 if GUARD then
   Result <= Arg1 + Arg2;
 end if;
end process LAB1;
```



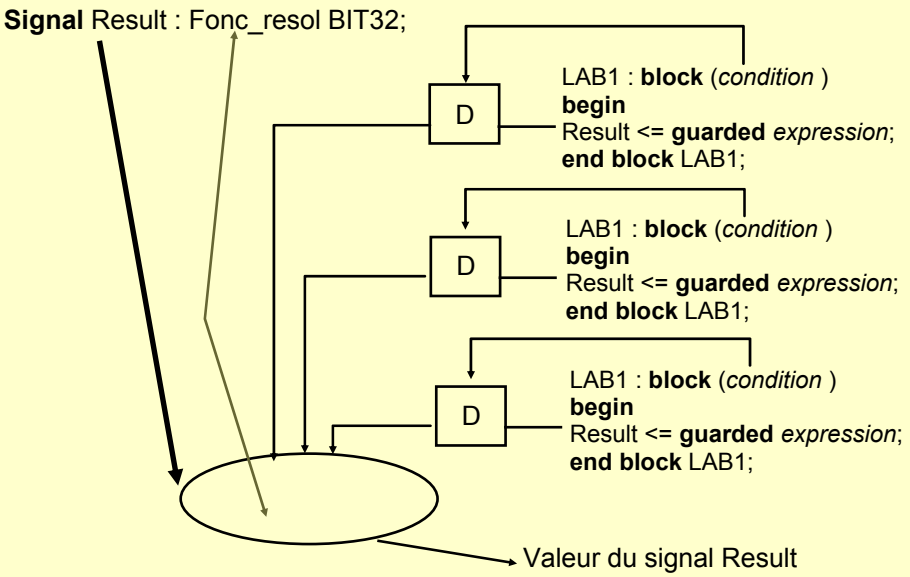
Autre exemple

```
signal A,B,S1,S2  VLBIT;

LAB1 : block (CLOCK = '1' )

begin
S1 <= A after 1 ns;
S2 <= guarded B after 1 ns;

end block LAB1;
```



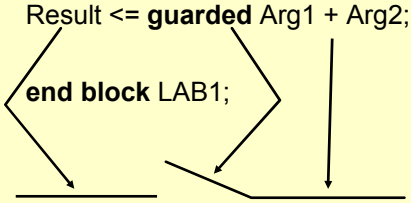
Remarque : Quand la condition de garde devient fausse, le pilote (driver) du signal est toujours actif et propose 0.

Blocs gardés et résolution



Signaux gardés de type "bus" et "register"

```
signal Result : Fonc_resol BIT32 bus;
signal Result : Fonc_resol BIT32 register;
LAB : block (not CLOCK'STABLE and CLOCK = '1' )
        --- front montant
begin
Result <= guarded Arg1 + Arg2;
end block LAB1;
```



Si la condition de garde devient fausse, on déconnecte le pilote du signal

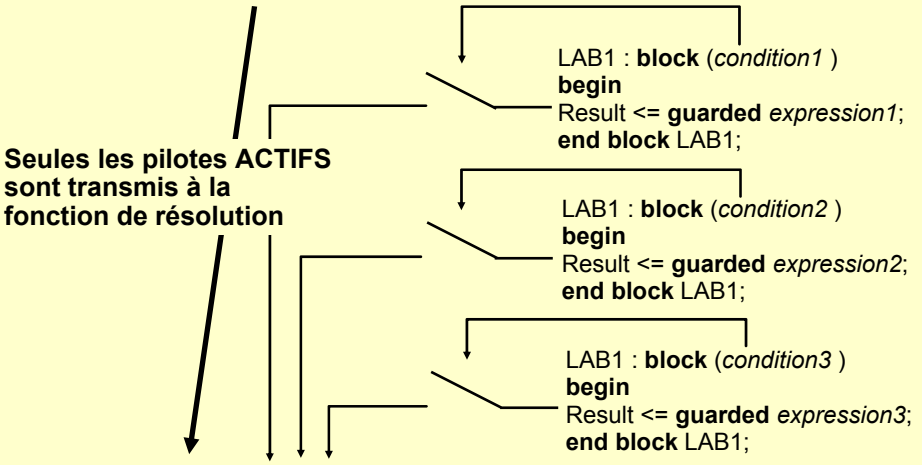
Bloc gardé (bus ou register) process équivalent

```
signal Result : Fonc_resol BIT32 bus;
signal Result : Fonc_resol BIT32 register;
LAB : block (not CLOCK'STABLE and CLOCK = '1' )
begin
Result <= guarded Arg1 + Arg2;
end block LAB1;
```

```
LAB1 : process (CLOCK)
begin
 if GUARD then
   Result <= Arg1 + Arg2;
 else
   Result <= null ; -- déconnexion
 end if;
end process LAB1;
```

Signaux gardés : Résolution

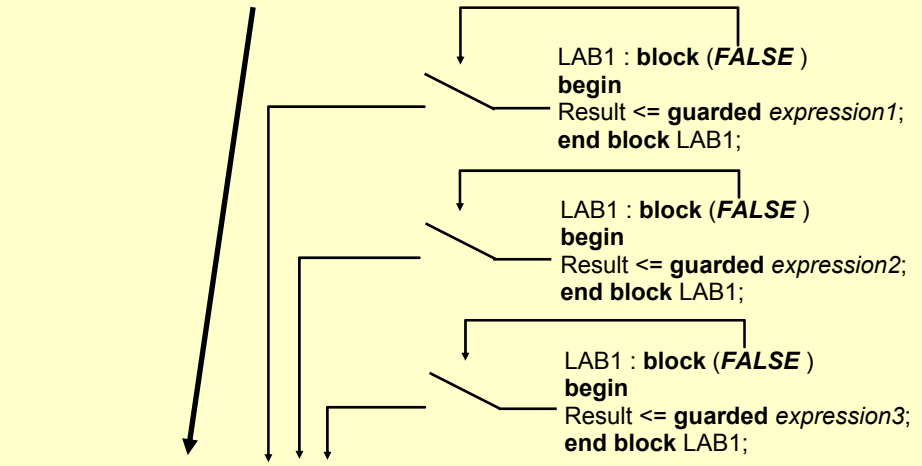
Signal Result : Fonc_resol BIT32 **bus**; (ou **register**)




```
function Fonc_resol (ARG : in BIT32_VECTOR) return BIT32 is
begin .....
```


Bloc gardé : Type bus

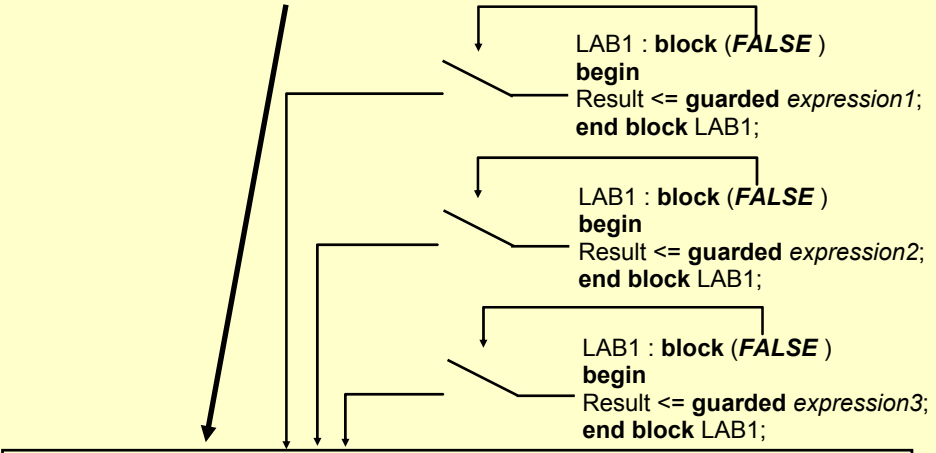
Signal Result : Fonc_resol BIT32 **bus**;




```
function Fonc_resol (ARG : in BIT32_VECTOR) return BIT32 is
begin ..... -- appelée même si toutes les conditions sont fausses
  if ARG'LENGTH = 0 then return valeur_appropriée -- ('Z')
```

Bloc gardé : Type register

Signal Result : Fonc_resol BIT32 **register**;



function Fonc_resol (ARG : **in** BIT32_VECTOR) **return** BIT32 **is**
begin .....
--n'est pas appelée si toutes les conditions sont fausses. La dernière valeur
-- affectée reste sur le signal

Signaux VHDL

Une seule source possible (Instruction d'affectation)

Signaux résolus

Plusieurs sources possibles (Plusieurs affectations)

Signaux gardés
(Bus ou Register)

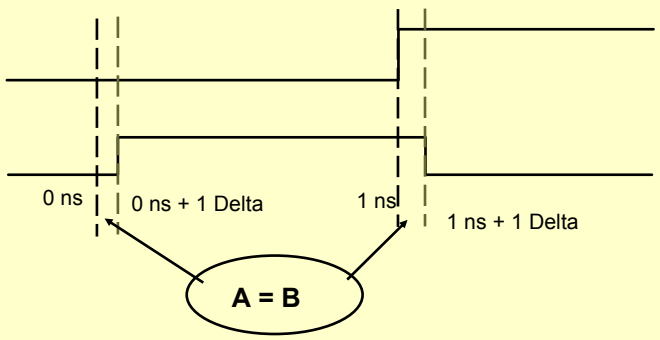
Instructions d'affectations gardés

Standard VHDL'92
Nouvelles Fonctionnalités

- Postponed
- Variables Globales
- Instantiation Directe
- Liaison incrémentale
- Groupes
- "Foreign Interfaces"
- Fichiers (lecture et écriture)
- Fonctions impures
- Opérateurs décalage et rotation
- Opérateur xnor
- Attribut prédéfini DRIVING_VALUE et 'DRIVING
- Affectation de signal inertiel
- Affectation d'expression aux ports d'entrée
- Affectation de signal, "unaffected"

Postponed (1)

```
A <= '0' , '1' after 1 ns;
B <= not A;
assert (B = not A) report "erreur ? éventuellement fausse alarme ";
```



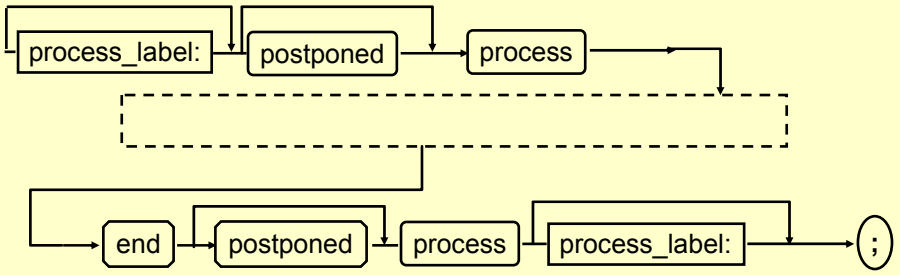
Postponed (2)

Pas d'affectation à délai 0
Pas d'attente nulle (0 ns)

Activation à la fin du Delta de Simulation

* Process
* Instruction ASSERT
* Appel concurrent de procédure
* Affectation de signal (Conditionnelle ou sélective)

postponed assert (B = **not** A) **report** "erreur réelle dans état stable";



Variables Globales ("Shared")

shared variable MA_VARIABLE **:** MON_TYPE **:=** INIT_VALUE;

Position : Toute partie déclarative sauf sous-programmes et process

Attention à l'Utilisation

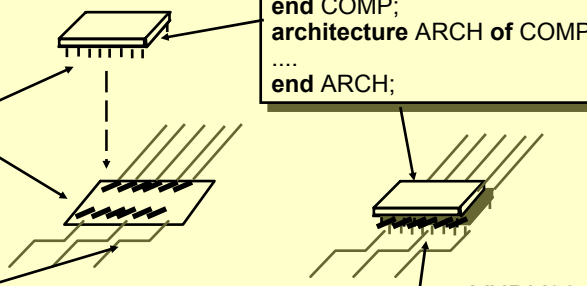
```
P1: process
begin
        COUNT:=1;
..
end process;

P2: process
begin
        COUNT:=2;
..
end process;
```

Instantiation Directe (1)

VHDL'87

```
entity TEST is
end;
architecture A of TEST is
component B generic(...);
                port(...);
end component B;
for C1:B use entity
          WORK.COMP(ARCH);
signal E1,E2,... : Same_type;
begin
C1:B generic map(...);
          port map(E1,E2,...);
....
end A;
```

```
entity COMP generic(...);
                port(...);
end COMP;
architecture ARCH of COMP is
....
end ARCH;
```

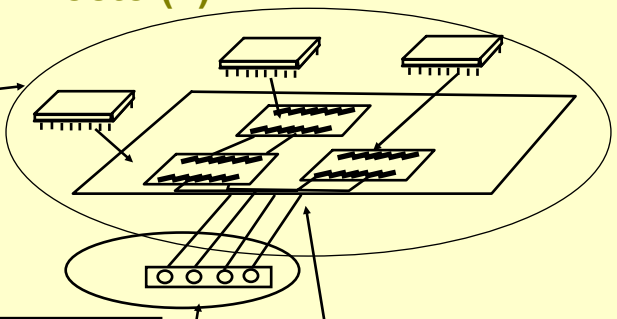


VHDL'92

```
entity TEST is
end;
architecture A of TEST is
signal E1,E2,... : Same_type;
begin
C1:entity WORK.COMP(ARCH)
          generic map(...);
          port map(E1,E2,...);
....
end A;
```


Instantiation Directe (2)

```
configuration CONF
          of COMP is
for ARCH
          ...
end for;
end CONF;
```



```
entity TEST is
end;
architecture A of TEST is
signal E1,E2,... : Same_type;
begin
C1:configuration WORK.CONF
          generic map(...);
          port map(E1,E2,...);
....
end A;
```

```
entity COMP generic(...);
                port(...);
end COMP;
architecture ARCH of COMP is
....
end ARCH;
```

Liaison incrémentale (1)

VHDL'87 --> generic et port map, unique pour chaque instance
Méthodologie avec rétro-annotation "timings"
Spécification de configuration dans l'architecture (instantiation) avec insertion des "timings" dans la même spécification de configuration et non plus tard, i. e. dans la déclaration de configuration.
VHDL'92 introduit cette liaison en deux étapes (liaison incrémentale)

Configuration de composant (dans déclaration de configuration)

```
for specification_de_composant
        [binding_indication;]
        [block_configuration]
end for;
```

binding_indication ::= [**use entity**_aspect] [**generic_map**_aspect] [**port_map**_aspect]

entity_aspect peut être omis si l'instance est déjà configurée dans une ou plusieurs spécification de configuration.

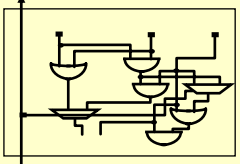
Spécification de configuration (architecture ou partie déclarative de block)
for component_specification binding_indication;

Liaison incrémentale (2)

```
entity AND3_ENT is generic(DELAY:TIME); port(A,B,C : IN BIT; S:OUT BIT);
end AND3_ENT;
architecture DATAFLOW of AND3_ENT is begin   S <= A and B and C after DELAY;
end DATAFLOW;

entity EXAMPLE is end;
architecture STRUCTURE of EXAMPLE is
        component AND2_COMP is port(X,Y: in BIT;Z:out BIT); end component;
for CI : AND2_COMP use entity WORK.AND3_ENT(DATAFLOW)
                        generic map(DELAY => 2.0 ns)
                        port map (A=>X, B=>Y, C=>open, S=>Z);
signal S1,S2,S3 : BIT;
begin  CI:AND2_COMP port map(S1,S2,S3);
end STRUCTURE;

configuration INCREMENTAL of EXAMPLE is
        for STRUCTURE
                for CI:AND2_COMP   generic map (DELAY=> 1.83 ns) -- remplace 2.0 ns
                                   port map (C=>'1'); --- précédemment non connecté
                end for;
        end for;
end INCREMENTAL;
```

Liaison incrémentale (3)

Conception de l'architecture



```
use WORK.BASIC_COMPONENTS.all;
architecture WITH_UNIT_DELAY of LOGIC_ENTITY is
 for all: NAND2 use entity CMOS.NAND2(BEH) generic map(2 ns);
 for all: NAND3 use entity CMOS.NAND3(BEH) generic map(3 ns);
...
begin
< Instructions d'instantiation de composants>
end WITH_UNIT-DELAY;
```

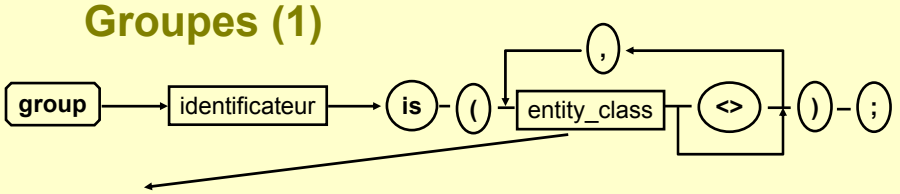
Simulation Fonctionnelle (délais estimés)

Layout → **Rétro-annotation**

```
Configuration WITH_EXACT_TIMING of LOGIC_ENTITY is
        for WITH_UNIT_DELAY      for CN21: NAND2 generic map(1.95 ns); end for;
                                 for CN22: NAND2 generic map(1.95 ns); end for;
                                 for CN31: NAND3 generic map(1.95 ns); end for;
                                 ...
        end for;
end WITH_EXACT_TIMING;
```

Simulation Logique (délais après rétro-annotation)

Groupes (1)



entity | architecture | configuration | procedure | fonction | package | type | subtype | constant | signal | variable | component | label | literal | units | group | file

group PATH **is** (**signal**,**signal**); -- Déclaration

group A_TO_S : PATH (A,S);

attribute PROPAGATION_DELAY **of** A_TO_S: **group is** 250 ns;

attribute PROPAGATION_DELAY : TIME;

Groupes (2)

```
entity CHIP is
        port(INPU1,INPUT2 : BIT; OUTPUT1, OUTPUT2 : BIT);
end CHIP;

group INPUT1_TO_OUTPUT1 : PATH(INPUT1,OUTPUT1);
group INPUT1_TO_OUTPUT2 : PATH(INPUT1,OUTPUT2);
group INPUT2_TO_OUTPUT1 : PATH(INPUT2,OUTPUT1);
group INPUT2_TO_OUTPUT2 : PATH(INPUT2,OUTPUT2);
constant DELAY:TIME:=50 ns;

attribute PROPAGATION_DELAY of INPUT1_TO_OUTPUT1: group is DELAY;
attribute PROPAGATION_DELAY of INPUT1_TO_OUTPUT2: group is DELAY;
attribute PROPAGATION_DELAY of INPUT2_TO_OUTPUT1: group is DELAY;
attribute PROPAGATION_DELAY of INPUT2_TO_OUTPUT2: group is DELAY;
```

factorisation par : **attribute** PROPAGATION_DELAY **of all:group is** DELAY;

Groupes (3)

De manière plus efficace :

```
group PIN_SET is (signal <>);
group PATHS is (group <>);

group SOURCES : PIN_SET(INPUT1,INPUT2)
group TARGETS : PIN_SET(OUTPUT1,OUTPUT2);

group INPUTS_TO_OUTPUTS : PATHS(SOURCES,TARGETS);
attribute PROPAGATION_DELAY of INPUTS_TO_OUTPUTS : group is DELAY
```

"Foreign Interfaces"

Interface avec le monde extérieur : -système hôte
- utilisateur (entrées/sorties)
- Autres logiciels (assembleur, ...)

Attribut FOREIGN (Package STANDARD) **attribute** FOREIGN:STRING

Ex: **function** GESTION_MENU(ARG: **in** MENU) **return** ITEM;
attribute FOREIGN **of** GESTION_MENU:**function is** "W$MENU";

Note : W$MENU primitive écrite et édition de lien à définir

Fichiers (lecture et écriture)

Modélisation et fichiers : - chargement de ROM ou PLA
- contrôle, monitoring du modèle

VHDL'92 le type **FILE** est une classe d'objets
utilisable dans une procédure :
procedure ADHOC(**file** F:TEXT)

procédures FILE_OPEN & FILE_CLOSE

déclaration **open** (pour input & output)

Fonctions impures

VHDL'87 - paramètres **in**
- pas d'effets de bord (impossible d'avoir une fonction PULL retournant le sommet d'une pile décrémentant le pointeur, pas de retour de caractère suivant pour les fichiers...)

VHDL'92

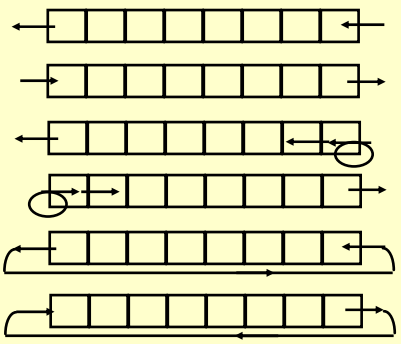
impure function F(V:INTEGER) **return** INTEGER;
(accès à un fichier externe possible, paramètres toujours **in**)

Orthogonalité : **pure** mot réservé, par défaut pour les autres fonctions

Opérateurs décalage et rotation

VHDL'87 : aucun

VHDL'92: tableaux à une dimension, forme: <type_tab> Opérateur INTEGER



sll (Shift Left Logical)
srl (Shift Right Logical)
sla (Shift Left Arithmetic)
sra (Shift Right Arithmetic)
rol (ROtate Left logical)
ror (Rotate Right logical)

Valeur de remplissage = type'LEFT ('0' pour BIT, FALSE pour booléen)
STD_LOGIC_1164 : opérateurs sur STD_ULOGIC

Opérateur xnor

VHDL'87 : **and, or, nand, nor, xor**

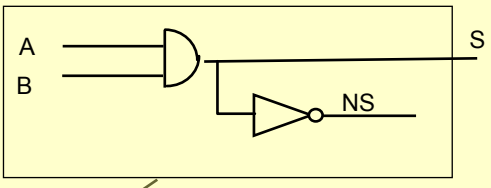
VHDL'92: **and, or, nand, nor, xor, xnor**

A **xnor** B équivalent à **not** (A **xor** B)

Types BIT ou BOOLEEN ou vecteurs

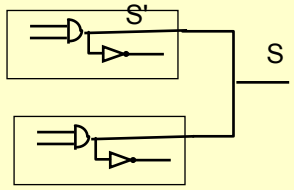
Attribut prédéfini DRIVING_VALUE et 'DRIVING

VHDL'87 : pas de lecture des ports de sortie



port(....;S: **out** BIT);

~~S <= A **and** B;~~
~~NS <= **not** S;~~



Cause : Drivers et résolution

Attribut DRIVING_VALUE

Utilisable dans tout process pilotant le signal (process ou instruction concurrente ayant un process équivalent) ou dans toute procédure ayant le signal en paramètre **out** ou **inout**)
Retourne la contribution du process ou de la procédure pour le signal, (erreur si pas de pilote ou pilote déconnecté)

Partie séquentielle :

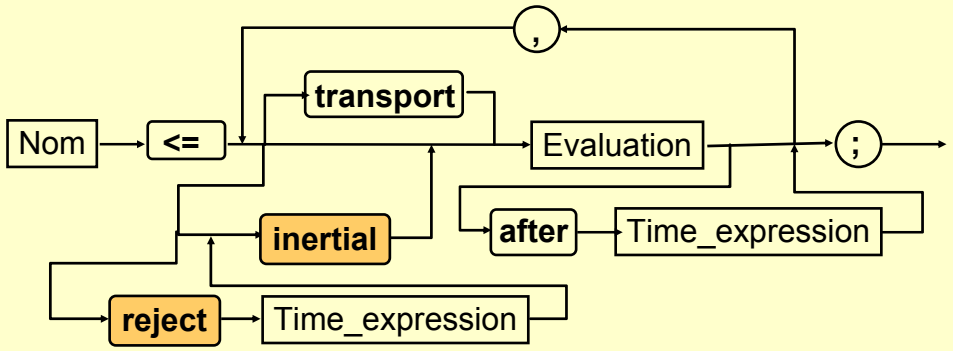
```
S <= A and B;
NS <= not S'DRIVING_VALUE;
```

Attribut DRIVING

Retourne la valeur FALSE si le pilote est déconnecté, permet d'éviter l'erreur d'appel de DRIVING_VALUE

```
if S'DRIVING
        then V := S'DRIVING_VALUE.
end if;
```


Affectation de signal inertiel (1)

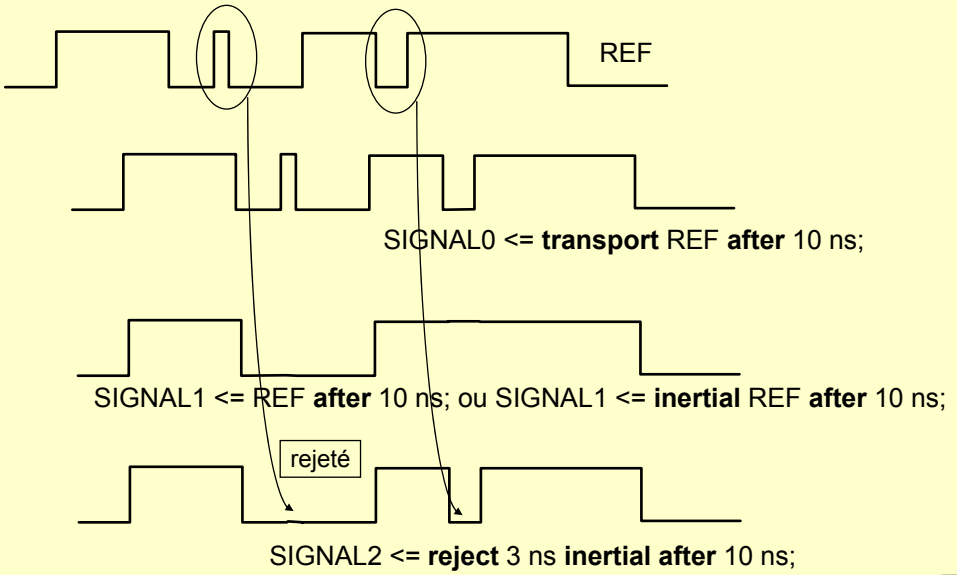



```
SIGNAL1 <= reject 3 ns inertial after 10 ns;
```

est identique à :

```
SIGNAL2 <= REF after 3 ns;
SIGNAL1 <= transport SIGNAL2 after 7 ns;
```

Affectation de signal inertiel (2)



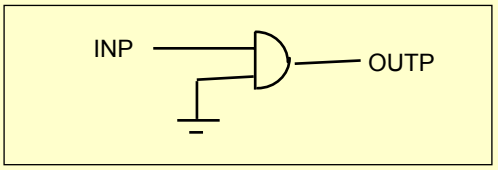
Affectation d'expression aux ports d'entrée (1)

VHDL'87

```
component COMP
port(I1,I2:in BIT:='0'; S: out BIT);
-- := nécessaire pour port open
end component;
...
signal INP,OUTP:BIT;
...
begin
C1:COMP port map(INP,open,OUTP);
...
```

VHDL'92

```
component COMP is
port(I1,I2:in BIT; S: out BIT);

end component COMP;
...
signal INP,OUTP:BIT;
...
begin
C1:COMP port map(INP,'0',OUTP);
...
```



Affectation d'expression aux ports d'entrée (2)

```
component JK is
        port (J,K,CLOCK: in BIT; Q,NQ : out BIT);
end component JK;


begin
DIVID_BY_2: JK port map (       J => '1';
                                K => '1';
                                CLOCK => INPUT;
                                Q => DIVID_2_OUTPUT;
                                NQ => open);
...
```

Permet d'éviter la déclaration de signaux pour les forçages de nature différente.
(VHDL'87 : une seule valeur par défaut dans la déclaration de port)

Affectation de signal, "unaffected"

```
VHDL'87 :   S <= '1' after 10 ns when CK='1' else S;
            -- éventuels problèmes logiques
solution    process     begin if CK='1' then
                                S<= '1' after 10 ns;
                                end if;
                        wait on CK;
            end process;
```

```
VHDL'92 :   S <= '1' after 10 ns when CK='1' else unaffected;
            S <= '1' after 10 ns when CK='1'; -- équivalent (else devient optionel)

            with EXP select
                    S <= '1' after 10 ns when ADD,
                    S <= '0' after 10 ns when SUB,
                    unaffected when others;
```